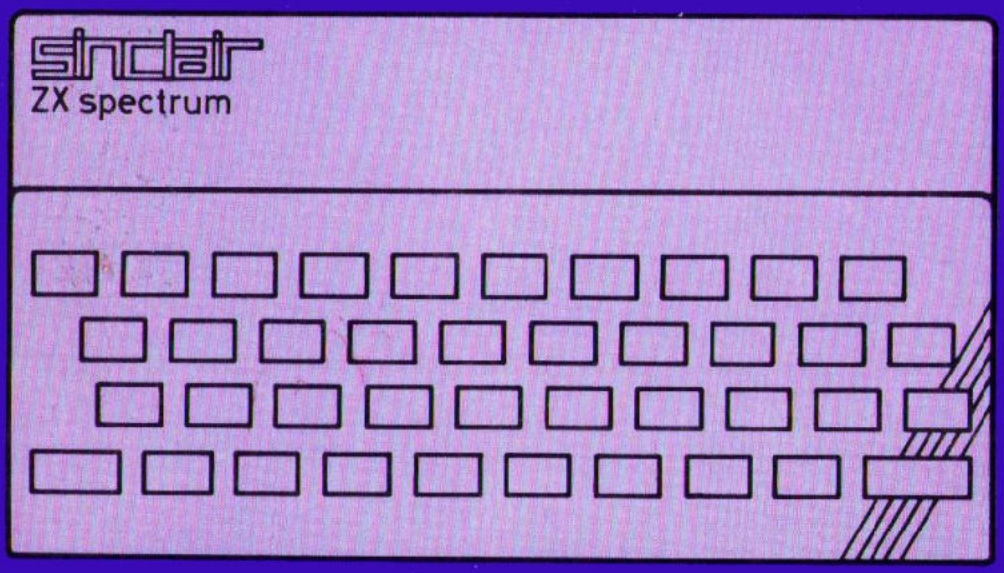
# ZX Spectrum

# LEREN PROGRAMMEREN




M. JAMES

DE MUIDERKRING

# ZX SPECTRUM

Oorspronkelijke titel: The Art of Programming the ZX Spectrum
Vertaling: Jos Verstraten



ISBN 90 6082 245 5

**M. JAMES**

**Vertaald door Jos Verstraten**

# ZX SPECTRUM

## LEREN PROGRAMMEREN

**DE MUIDERKRING B.V. – BUSSUM**

UITGEVERIJ VAN TECHNISCHE BOEKEN EN TIJDSCHRIFTEN

# INHOUD


Voorwoord ..... 7

1. Leer uw Spectrum kennen ..... 8

2. Grafische mogelijkheden
met lage resolutie ..... 10

3. Spelen met toeval ..... 20

4. Grafische mogelijkheden
met hoge resolutie ..... 29

5. Geluid ..... 37

6. Bewegende beelden ..... 45

7. PEEK en POKE ..... 54

8. Gevoel voor tijd ..... 60

9. Strings en woorden ..... 64

10. Grafische mogelijkheden
voor gevorderden ..... 70

# VOORWOORD

Haalt u voor het eerst van uw leven een computer uit de verpakking? Of bent u reeds een fanatiek programmeur? Het maakt niets uit, vragen als "hoe kan ik dit of dat probleem op mijn nieuwe Spectrum programmeren" zullen in beide gevallen door uw hoofd spoken.
Dit boek is bedoeld om op sommige van die vragen een antwoord te geven. Het wil u helpen bij het ontwikkelen van programmeertechnieken, waarmee u simpele maar zeer effectieve BASIC-programma's kunt schrijven.
In deze uitgave worden alle mogelijkheden van de Spectrum, die u nodig heeft bij het schrijven van programma's voor leuke en opwindende spelletjes, behandeld. Zo komen in hoofdstuk 2 de mogelijkheden van kleurige grafieken met lage resolutie aan de orde. In het volgende hoofdstuk zullen we de geheimen van de "generator van willekeurige cijfers" (random number generator) verklappen. Hoofdstuk 4 gaat in op grafieken met hoge resolutie. In hoofdstuk 5 gaan we op ontdekkingstocht naar de geluidsbronnen van uw machine. "Bewegende beelden" luidt de veelbelovende titel van het volgende hoofdstuk, terwijl nummer zeven zich bezig houdt met twee wel zeer krachtige instrukties: PEEK en POKE. Hoofdstuk 8 is gewijd aan de interne tijdmechanismen van de machine en in hoofdstuk 9 gaan we in op de "string"-mogelijkheden van onze computer. In het laatste hoofdstuk zetten we nogmaals de grafische mogelijkheden van de Spectrum in het zonnetje, maar nu op een wel erg hoog niveau.
Een boek heeft een begin en altijd een einde. Denk echter niet dat het einde van dit werkje ook het einde betekent van het spelen met uw computer! Integendeel: de Spectrum heeft in feite onuitputtelijke mogelijkheden.
Laat deze uiteenzetting u op weg helpen bij het zelfstandig zoeken naar die oneindige programmerings-capaciteiten van uw Spectrum-computer!

M. James

# 1. LEER UW SPECTRUM KENNEN

De Sinclair Spectrum heeft zijn gebruikers erg veel te bieden. U kunt kleurige prentjes op het scherm van uw TV-toestel tekenen, zowel met hoge als met lage resolutie. U kunt deze grafiekjes bovendien laten begeleiden door allerlei geluiden. Dit alles als reslutaat van effectieve en spannende programma's, in BASIC geschreven. Alleen, u moet natuurlijk wél weten hoe dat moet! Het probleem is dat er toch wel iets meer bij komt kijken dan zo op het eerste gezicht lijkt. Het is namelijk niet voldoende dat u alle BASIC-instructies uit uw hoofd kent en weet wat ze betekenen. U moet ze ook met elkaar kunnen combineren tot programma's die precies dat doen wat u wilt. Dát is de kunst en de fijne kneepjes van die kunst wil dit boek u leren.

**Een opwindend vooruitzicht**

In deze uitgave komen achtereenvolgens het grafiekjes tekenen met lage resolutie (hoofdstuk 2), werken met hoge resolutie (hoofdstuk 4) en het opwekken van geluiden (hoofdstuk 5) aan de beurt.

Nu bieden tegenwoordig een heleboel huiscomputers de mogelijkheid om met geluid en kleur te spelen. Wat is er dan toch zo speciaal aan de Spectrum, zult u vragen. Om te beginnen is het een groot voordeel dat we met de Spectrum zowel met lage als met hoge resolutie kunnen werken. Over het algemeen kunnen we sneller en gemakkelijker programmeren als we gebruik maken van de lage resolutie-faciliteiten. Maar er zijn afbeeldingen, zoals bijvoorbeeld fijne omtrekken en vormen, die alleen maar te realiseren zijn door te programmeren met hoge resolutie-methoden. Door beide systemen door elkaar in één programma te gebruiken, kunnen we een veel uitgebreidere set tekens en symbolen op het scherm zetten, dan met de meeste andere computers het geval is.

Ten tweede heeft de Spectrum een zeer gerieflijk kleurenpallet ter beschikking. Weliswaar zijn er enige beperkingen in het kleurengebruik bij hoge resolutieprentjes, maar bij het toepassen van lage resolutie-tekeningen kunnen we onze kleuren willekeurig en vrijelijk over het scherm uitsmeren.

Ten derde spreekt de Spectrum een BASIC-dialect dat een aantal erg nuttige extra's heeft (die komen in dit boek uiteraard aan bod). Het programmeren is erg eenvoudig gemaakt door het onder één toets brengen van alle sleutelwoorden. Dit dialect kan zelfs uw programmeringsfouten ontdekken op het moment dat u ze maakt. In de praktijk zal blijken hoe handig dat is.

**De kunst van het programmeren**

Dit boek wil u helpen bij het schrijven van uw eigen programma's. Het is zeer zeker niet bedoeld als een volledige introductie in de BASIC-taal of in de geheimen van de machine. Alvorens u verder leest, zult u het toetsenbord van de Spectrum moeten kennen om enige simpele BASIC-programmaatjes te kunnen schrijven.

We gaan ervan uit, dat u de handleiding van uw Spectrum heeft gelezen, weet hoe u de machine aan de praat moet krijgen en voldoende BASIC-kennis heeft voor het lezen en schrijven van korte programma's. Dat wil nog niet zeggen dat wij van u verlangen dat u uw ideeën kunt omzetten in bruikbare Spectrum-programma's. Programmeren vereist meer dan alleen maar de kennis van de BASIC-instructies, net zoals het begrijpen van een vreemde taal meer inhoudt dan alleen maar het kunnen uitspreken van een paar woorden.

Op dit punt gekomen, kunt u dit boek weer ter hand nemen. Het zal u vertrouwd maken met programmeringstechnieken en u leren hoe u uw ideeën kunt realiseren op de Spectrum. Met andere woorden: u kunt van dit boek verwachten dat het u leert hoe een programma uit diverse, tamelijk eenvoudige, stappen wordt opgebouwd en hoe u extra's kunt invoegen op het moment dat ze nodig zijn. Het bevat talloze deelprogramma's maar daarnaast ook een twintigtal volledig uitgewerkte programma's voor leuke spelletjes. Deze spelletjes vormen een welkome onderbreking in het leerproces en zijn ideaal om

aan uw vrienden te demonstreren hoe ver u bent gevorderd in de programmeringskunst. Natuurlijk zijn ook deze uitgewerkte programma's te verbeteren: er bestaat immers niet zoiets als een ideaal programma! Gebruik de spelletjes dus niet alleen als ontspanning: zet uw verbeelding aan het werk en probeer uw ideeën in de programma's te verwerken. Vaak blijkt het veel leerzamer van een bestaand programma uit te gaan en uw ideeën daarin te verwerken, dan met niets te beginnen.

**Leren en spelen**

Misschien verbaast u zich een beetje over het feit dat ook in deze uiteenzetting de nadruk wordt gelegd op programma's voor spelletjes. Bij oppervlakkig doorlezen van de inhoudsopgave vindt u misschien zelfs geen enkel "ernstig" programma. Een voor de hand liggende reden voor het ontwikkelen van spelletjesprogramma's is het feit dat we daarin alle mogelijkheden van de Spectrum kunnen verwerken. Daarnaast willen we ons op een aangename manier bezig houden met het leren programmeren. We leren sneller en gemakkelijker, als we plezier hebben in dat wat we doen! Vandaar dat het belangrijk is spelletjes niet naar het rijk van de nonsens te verbannen. De scheidingslijn tussen louter spel en de ware kennis is bij de computerwetenschap moeilijk te trekken. Vergeet niet dat de hedendaagse rage van computer-ruimtevaartspelletjes zijn wortels heeft in programma's die zijn ontwikkeld en gebruikt bij de voorbereidingen van de eerste bemande vlucht naar de maan!

**Uw Spectrum systeem**

Dit boek is geschreven aan de hand van een Spectrum met een geheugen van 16K, een doodgewone TV, een idem cassetterecorder en een ZX-printer voor de illustraties.
Iedere Spectrum, zowel de 16K als de veel krachtigere 48K, is dus bruikbaar voor het uitvoeren van alle in dit werkje beschreven programma's.

Hoeveel de Spectrum een kleurencomputer is, kan men ook een zwart-wit TV gebruiken. De diverse kleuren presenteren zich dan als tinten grijs. Toch zijn er enige kleurencombinaties, zoals bijvoorbeeld rood en groen, die niet zo duidelijk van elkaar te onderscheiden zijn op een zwart-wit toestel. Experimenteer dus maar wat en schakel over naar die kleuren die duidelijk contrasterend zijn.
De behandelde en/of uw eigen programma's, ingetoetst in uw computer, kunnen worden bewaard door ze op te slaan op een cassette of op een micro-floppy van de Sinclair ZX Microdrive.

# 2. GRAFISCHE MOGELIJKHEDEN MET LAGE RESOLUTIE

Hoewel de Spectrum gebruik maakt van slechts één systeem voor het presenteren van informatie op het scherm (de zogenoemde display mode), kunnen we de inhoud van het scherm op twee verschillende manieren wijzigen. Het systeem waarmee de computer tekst zowel als grafieken op het scherm schrijft is gelijk. De twee manieren waarmee we de inhoud van het scherm kunnen wijzigen, zijn in feite twee verschillende programmeringstechnieken voor het bereiken van een en hetzelfde doel: het veranderen van de beeldinformatie. Dit gebeurt ofwel door te programmeren met lage resolutie, ofwel door te programmeren met hoge resolutie. Daarbij is het belangrijk voor ogen te houden dat dit twee verschillende soft-ware (dus programmeringsgerichte) benaderingen zijn van een hard-ware (dus machnine-technisch) proces: het systeem waarmee de computer informatie op het scherm van een TV weergeeft.

**Het scherm van de Spectrum**

Het scherm van de Spectrum is samengesteld uit 256 horizontale en 176 vertikale punten (dots). Iedere punt kan ofwel ''aan'' ofwel ''uit'' worden geprogrammeerd. Een willekeurige grafische vorm (een letter, cijfer, symbool) wordt opgebouwd door een aantal punten volgens een bepaald patroon ''aan'' en ''uit'' te sturen. Nu moeten we nog exact definiëren wat we bedoelen met de begrippen ''aan'' en ''uit''. Laten we maar eens starten met twee kleuren en om de zaak zo eenvoudig mogelijk te houden, met zwart en wit. We kunnen dan stellen dat een ''aan''-punt zwart op het scherm wordt weergegeven en een ''uit''-punt wit. De vorm van een letter kan dan opgebouwd worden door ''aan''-punten te omringen met ''uit''-punten. De letter verschijnt dan zwart op het scherm, tegen een witte achtergrond. Dit systeem vertoont natuurlijk een grote gelijkenis met het met een zwarte pen schrijven op een wit vel papier. Vandaar dan ook, dat we vanaf nu spreken over ''inkt''-punten.
In het kort samengevat is het beeld op het TV-scherm opgebouwd uit twee verschillende soorten punten: inktpunten en papierpunten.
In de meeste gevallen zullen de inktpunten een andere kleur hebben dan de papierpunten.
De Spectrum heeft twee instrukties, waarmee we de kleur van de inkt- en de papierpunten kunnen programmeren:

```
INK 'kleurnummer'
```

en

```
PAPER 'kleurnummer'
```

Het woordje ''kleur'' staat hierbij voor een bepaald cijfer en ieder cijfer correspondeert met een bepaalde kleur voor zowel de papier- als inktpunten. Het is niet nodig het verband tussen cijfer en kleur uit het hoofd te leren. De kleuren staan immers genoteerd op de bovenste rij toetsen van het Spektrum-toetsenbord. Volledigheidshalve zetten we ze even op een rijtje.

0 - zwart
1 - blauw
2 - rood
3 - magenta
4 - groen
5 - cyaan
6 - geel
7 - wit.

De instruktie ''INK 2'' heeft tot gevolg dat alle inktpunten rood worden weergegeven, de instruktie ''PAPER 6'' levert een gele achtergrond op.
We kunnen de inkt- en papierkleuren volledig onafhankelijk van elkaar vaststellen. In wezen kunnen we zelfs de inkt- en papierkleur aan elkaar gelijk maken! De instruktie ''INK 4'', gevolgd door de instruktie ''PAPER 4'' heeft een volledig groen scherm tot gevolg. Toch moeten we ons realiseren dat er ook nu twee verschillende sooten punten op het scherm worden weergegeven, namelijk inkt- en papierpunten, die echter door de rare instrukties dezelfde kleur hebben.

De volgende vraag die we gaan beantwoorden, is hoe het mogelijk is meer dan twee kleuren tezelfdertijd op het scherm weer te geven. Het scherm is verdeeld in een groot aantal vierkanten, ieder opgebouwd uit 8 horizontale en 8 vertikale punten. Zo'n vierkant noemen we een "symboolplaats" (character location).
We kunnen in iedere symboolplaats de kleur van de inkt- en papierpunten onafhankelijk van elkaar vaststellen. Hoewel dit op het eerste gezicht een beetje vreemd lijkt, heeft dit wel tot resultaat dat we zeer eenvoudig kunnen programmeren op de Spectrum. De 64 punten in een symboolplaats zijn voldoende voor het duidelijk opbouwen van letters, cijfers en zelfs van de kleine en eenvoudige grafische symbolen, die vaak bij spelletjes worden gebruikt.
Het is dan ook veel overzichtelijker het scherm te beschouwen als een verzameling symboolplaatsen dan als een verzameling punten.
Dat is dan ook precies wat de "PRINT-instruktie doet. Als u een symbool door middel van een "PRINT"-instruktie op het scherm schrijft, dan bepaalt u in feite welke punten in een bepaalde 8 bij 8 symboolplaats papierpunten en welke inktpunten zijn. Bovendien kunnen we de kleur van de inkt- en de papierpunten voor iedere symboolplaats programmeren.
Door middel van de "PRINT"-instruktie kunnen we dus de inhoud van een symboolplaats wijzigen. Het opbouwen van het totale schermbeeld door wijzigen van de inhoud van de symboolplaatsen, noemen we programmeren met lage resolutie. De Spectrum heeft ook een instruktieset, waarmee we ieder punt van het scherm kunnen programmeren als inkt- of als papierpunt. Het gebruik van deze instrukties voor het opbouwen van een grafiek noemen we programmeren met hoge resolutie. Het enige probleem daarbij is dat we toch beperkt blijven tot het gebruik van slechts één inkt- en één papierkleur per symboolplaats. Vandaar dat het erg moeilijk is om door middel van hoge resolutie-programma's "full-colour"-plaatjes op het scherm op te bouwen.
De meeste programma's maken dan ook gebruik van de lage resolutie-instrukties. Deze techniek gaan we in de rest van dit hoofdstuk in detail behandelen, het gebruik van de hoge resolutie-instrukties bewaren we tot hoofdstuk 4.

**De "PRINT"-instruktie - komma, puntkomma, afkappingsteken**

De algemene vorm van een "PRINT"-instruktie luidt:

```
PRINT 'print list'
```

waarin "print list" staat voor een aantal gegevens die we graag geprint willen hebben. Die gegevens kunnen we van elkaar scheiden door komma's, puntkomma's of afkappingstekens. Deze leestekens hebben in de BASIC-taal een belangrijke functie en worden aangeduid met de term "seperators", scheiders.
Deze seperators bepalen de manier waarop de gegevens van een "PRINT"-instruktie op het scherm worden weergegeven. Zo zal het gebruik van puntkomma's tussen de gegevens van een een "PRINT"-bevel tot gevolg hebben dat deze informatie zonder spatie op één regel van het scherm wordt geschreven.

Een voorbeeldje:

```
PRINT "a";"b"
```

schrijft ab op het scherm.
De functie van de drie separators kan als volgt worden samengevat:

— puntkomma:
de printgegevens worden achter elkaar zonder spatie geschreven;

— komma:
schrijft ieder gegeven van het bevel in een nieuwe printzone;

— afkappingsteken:
schrijft ieder gegeven op een nieuwe lijn van het scherm.

Het begrip "printzone" gaan we nu even toelichten. Het scherm van de Spectrum bestaat horizontaal uit 256 punten, verdeeld in 32 symboolplaatsen van ieder 8 punten. In feite is het scherm opgebouwd uit 32 kolommen en die 32 kolommen zijn opgesplitst in twee "printzones". De eerste zone loopt van kolom 1 tot en met kolom 16, de tweede gaat van kolom 17 tot en met kolom 32.

Uit de instruktie:

```
PRINT "a","b"
```

vloeit voort dat de letter a in het begin van een regel wordt afgedrukt en de letter b in de 17-de kolom van die regel.
Er valt nog één belangrijk feit over de separators op te merken. Het eindigen van een "PRINT"-bevel met gelijk welke separator heeft tot resultaat dat de normale procedure, namelijk het naar een nieuwe regel gaan op het einde van de "PRINT", wordt onderdrukt.
Met andere woorden:

```
10 PRINT "a"
20 PRINT "b"
```

plaatst de letters a en b op twee verschillende regels.

```
10 PRINT "a",
20 PRINT "b"
```

heeft echter hetzelfde effect als:

```
10 PRINT "a","b"
```

namelijk de letters a en b op één regel, maar in twee verschillende zones.
Door het afsluiten van een "PRINT"-instruktie met een afkappingsteken, zal de computer de daaropvolgende "PRINT"-instruktie rechtstreeks, zonder spatie, achter de eerste instruktie op het scherm schrijven.
Een voorbeeld:

```
10 PRINT "a";
20 GOTO 10
```

vult het scherm met a's.
Het feit dat het afsluiten van een regel met een separator het automatische "naar de volgende regel"-mechanisme onderdrukt, heeft ook vreemde gevolgen.
Dit wordt bijvoorbeeld aangetoond door:

```
10 PRINT "a";
```

Het beëindigen van een "PRINT"-instruktie met een afkappingsteken is de oorzaak dat de computer de cursor naar de start van de volgende lijn stuurt. Een daaropvolgende "PRINT"-instruktie stuurt de cursor dan weer naar de start van een nieuwe lijn: er wordt een witregel gevormd!
Het intikken van de instruktie "PRINT" . . ." geeft dus één witregel, het intikken van "PRINT" . . ." geeft twee witregels.

**De "PRINT TAB"- en "PRINT AT"-instructies**

Door een zorgvuldige toepassing van de komma- en de puntkomma-separators kunnen we de meeste problemen bij het indelen van onze tekst de baas. Toch is het vrij ingewikkeld om, als we alleen maar de beschikking hebben over deze separators, een symbool op een bepaalde plaats op het scherm te zetten. Dit probleem kunnen we oplossen door gebruik te maken van de "TAB"-instructie.
Het opnemen van de instructie "TAB(N)" in een "PRINT"-regel heeft tot gevolg dat het daaropvolgende teken verschijnt in kolom N van die regel. Indien echter de regel al tot na die N- de kolom was volgeschreven, zal de computer de tekst na de "TAB(N)"-instructie op de N-de kolom van de volgende regel weergeven. Dit wordt aangetoond aan de hand van het volgende programma:

```
10 PRINT TAB(25);"ab";TAB(25);"ab"
```

Noteer, dat het geen nut heeft een komma-separator op te nemen achter het "TAB"-bevel. De schrijfpositie zou daardoor alleen maar verschuiven naar waar de TAB hem had geplaatst!
Indien u aan N een grotere waarde dan 32 toekent, zal de computer zoveel maal het getal 32 van die waarde aftrekken, tot het resultaat in het bruikbare gebied valt. Op die plaats zal de computer de op het "TAB"-bevel volgende gegevens plaatsen.
Alle tot nu toe behandelde "PRINT"-bevelen beperken het verplaatsen van tekst tot de lijn waarop de computer aan het schrijven is. Er is een bevel, namelijk de "PRINT AT"-instructie, waarmee we gelijk wat op elke willekeurige plaats van het scherm kunnen zetten.
Dit "AT"-bevel is zeer gemakkelijk in het gebruik.

```
PRINT AT Y,X;"WOORD"
```

zal de tekst "WOORD" in kolom X van regel Y schrijven. Het maakt niets uit of er al iets op

die plaats op het scherm stond. Dat wordt gewoon vervangen door de nieuwe tekst. Zoals reeds gezegd is het scherm van de Spectrum opgebouwd uit 22 regels met ieder 32 symbolen. De eerste regel bovenaan het scherm heeft als volgnummer 0. De eerste kolom aan de linkerzijde van het scherm heeft eveneens 0 als volgnummer. Hieruit volgt dat aan X een waarde tussen 0 en 31 kan worden toegekend en aan Y een waarde tussen 0 en 21. Geeft men één van beide gegevens een te hoge waarde, dan reageert de computer door op het scherm de foutmelding "Integer out of range" te schrijven. Een eenvoudig voorbeeldje van de "PRINT AT"-instruktie is:

```
10 PRINT AT 11,16;"*"
```

Dit programmaatje schrijft een sterretje in het midden van het scherm. Op dezelfde manier kunnen we het "PRINT AT"-bevel gebruiken voor het tekenen van eenvoudige vormen op het scherm. Het resultaat van bijvoorbeeld:

```
10 FOR i=0 TO 31
20 PRINT AT 10,i;"*"
30 NEXT i
```

```
40 FOR i=0 TO 21
50 PRINT AT i,15;"*"
60 NEXT i
```

is, dat er twee lijnen (een horizontale en een vertikale) op het scherm worden getekend, opgebouwd uit sterretjes.
In een en hetzelfde "PRINT"-bevel kan men zoveel AT-instrukties opnemen als men zelf wil.

**Grafische symbolen**

Met vorige voorbeeld is aangetoond hoe we met behulp van het "PRINT AT"-bevel vormen op het scherm kunnen schrijven, in plaats van één symbool op één vaste plaats. Natuurlijk zijn lijnen en vormen, opgebouwd uit sterretjes, nu niet bepaald te omschrijven als mooie grafische voorstellingen. De spectrum beschikt echter over een aantal speciale grafische symbolen. Van de totale beschikbare set kunnen er 21 door de gebruiker van de computer zelf worden gedefiniëerd. Deze komen later aan de orde. Daarnaast zijn er 16 symbolen, die vast geprogrammeerd zijn en die zonder problemen kunnen worden gebruikt voor het tekenen van grafische vormen.
Deze zestien grafische symbolen vindt u op de

bovenste rij toetsen van het toetsenbord. Of, meer waarheidsgetrouw, u vindt daar 8 symbolen. De overige acht zijn de "inverse" van de getekende, dat wil zeggen dat de vorm identiek is, maar wit wordt zwart en omgekeerd.
U kunt over deze grafische symbolen beschikken door de computer naar grafiek om te schakelen. U doet dat door het gelijktijdig indrukken van de toetsen "CAPS SHIFT" en "GRAPHICS". De computer reageert onmiddellijk: de aan en uit flikkerende letter van de cursor op het scherm wordt vervangen door een "G" (van graphics). Het indrukken van één van de toetsen waarop de grafische symbolen zijn afgedrukt, laat dit symbool op het scherm verschijnen.
De inverse symbolen zijn op te roepen door het gelijktijdig indrukken van de symbooltoets en de "CAPS SHIFT"-toets.
Wij raden u aan om nu het lezen even te staken en met deze grafische mogelijkheden te gaan spelen. Het is belangrijk dat we voor ogen houden dat de inverse grafische symbolen, voor de computer, heel andere symbolen zijn dan de niet-inverse. Dit lijkt duidelijk, maar verderop zullen we een methode bespreken, waarmee we ieder symbool kunnen omzetten in zijn inverse.
Het weer naar gewoon schrift omschakelen, volgt door het nogmaals indrukken van de "GRAPHICS"-toets. Het is vrij moeilijk om de grafische symbolenset van de Spectrum op te nemen in deze tekst, omdat de zetmachines die de tekst van dit boek zetten niet over deze symbolen beschikken. Vandaar dat we haakjes gebruiken rond het cijfer of de letter van de toets die moet worden ingedrukt om een bepaald grafisch symbool op het scherm te laten verschijnen. Als in de tekst staat [3], dan bedoelen we het grafische symbool dat op het scherm verschijnt als de computer op grafisch gebruik is geschakeld en toets 3 wordt ingedrukt. Daarnaast gebruiken we in de tekst het teken om aan te duiden dat ook de toets "CAPS SHIFT" moet worden ingedrukt. [^3] is dus de code, die we in dit boek bezigen als we het grafisch symbool bedoelen dat ontstaat als de toetsen "3" en "CAPS SHIFT" gelijktijdig worden ingedrukt.
Natuurlijk kunnen we met de 16 standaard grafische symbolen lang niet alles op het scherm tekenen wat in onze verbeelding kan ontstaan. Wél zijn deze symbolen erg nuttig om eenvoudige vormen en figuren weer te geven bij de verschillende spelletjes.
Als u bijvoorbeeld een hondje op het scherm wilt zien, tik dan het volgende programma in:

```
10 PRINT "[1] [^1] [^3]""[8] [^5] [5]"
```

Hoe we dit hondje over scherm kunnen laten bewegen, is een verhaal dat we in hoofdstuk 6 zullen vertellen. Het spelen met de eigen grafische symbolen van de Spectrum is wel leuk, maar gaat snel vervelen. Bovendien is de praktische bruikbaarheid vrij klein. Voor het tekenen van vormen zoals rechthoeken of cirkels kunt u veel beter gebruik maken van programmeertechnieken met hoge resolutie.
Ook het op het scherm tekenen van figuurtjes zoals hondjes kunt u beter niet doen met de 16 beschreven grafische symbolen, maar met de later te bespreken symbolen, die door uzelf te definiëren zijn, de zogenoemde "user-defined"-symbolen.
Toch is het steeds de moeite waard even na te denken of een bepaald prentje niet is op te bouwen met de zeer eenvoudig toegankelijke vaste grafische symbolen. Deze zijn zeer geschikt voor het tekenen van dikke horizontale of verticale lijnen.
Toets maar eens het volgende programma in.

```
10 FOR i=1 TO 32
20 PRINT "[3]";
30 NEXT i
40 FOR i=1 TO 31
50 PRINT TAB(16);"[5]"
60 NEXT i
```

Let op het gebruik van de "TAB"-instruktie en van de puntkomma-separator, die de plaats van de lijnen op het scherm bepalen.

**De "CHR$-instruktie**

Naast de tot nu toe beschreven methoden, biedt de Spectrum nóg een manier voor het produceren van grafieken of gelijk welke andere symbolen: de "CHR$"-functie.
Als u alle symbolen, letters en cijfers die door de Spectrum kunnen worden geschreven, onder elkaar zet en ze opeenvolgend nummert, dan kunt u één symbool uit dat rijtje definiëren door te zeggen "ik bedoel het 36ste symbool".
Dat is precies wat de "CHR$"-functie doet. CHR$ (36) is de computernotatie voor het 36ste

symbool uit de totale symbolenset van de machine.
U kunt die volledige symbolenset op het scherm laten uitschrijven, door het volgende programma in te toetsen.

```
10 FOR i=32 TO 255
20 PRINT CHR$(i);
30 FOR j=1 TO 100
40 NEXT j
50 NEXT i
```

U zult opmerken dat soms een enkel symbool wordt uitgeprint en soms een volledig woord, zoals "COPY" of "PRINT". De Spectrum behandelt alle BASIC-instructies als enkele symbolen: de instrukties zijn immers op te roepen door één enkele toetsdruk.
De belangrijkste eigenschap van de "CHR$"-functie is, dat hij een rechtstreeks verband legt tussen getallen en symbolen. Als voorbeeld kunt u het volgende programma intoetsen en laten uitprinten.

```
10  PRINT CHR$(INT(RND*16+128));
20  GOTO 10
```

Het scherm wordt volgeschreven met symbolen op een willekeurige manier: er bestaat nu geen verband tussen de volgorde van de symbolen. Hoe dit programma precies werkt, wordt verklaard in het volgende hoofdstuk, waar we ingaan op de werking en het gebruik van de "RDN"-functie. Toch wordt door dit programma zeer duidelijk aangetoond hoe de "CHR$"-functie een verband legt tussen getallen en symbolen. Laat hetzelfde programma maar eens lopen zonder de "CHR$"-functie!

**Zelf te definiëren grafische symbolen**

Even een opmerking van de vertaler: er zijn korte Engelse computertermen, die een mond vol vertaling opleveren. "User-defined" is er zo een. Vandaar dat we in dit boek de term "U-D"-symbolen zullen introduceren.
Zonder enige twijfel haalt de lage resolutiemethode zijn grootste kracht uit het feit dat we in staat zijn zélf grafische symbolen te definiëren. Dat kan vrij eenvoudig en zal de nauwkeurigheid en gedetailleerdheid van onze tekeningen vergroten. Bovendien kunnen we de nieuwe, zelf ontworpen, grafische symbolen net zo gemakkelijk toepassen als de eigen symbolen van de computer.

Blijft de vraag waar die nieuwe symbolen verscholen zitten. Als u de machine in de grafiekstand zet en een lettertoets tussen de A en de U indrukt, dan verschijnt (vreemd genoeg) die letter op het scherm. Het lijkt dus alsof er twee manieren zijn om die letters op het scherm te krijgen. De "gewone" werkwijze van de machine en de grafiekstand. Dat komt omdat de ontwerpers van de machine in beide gevallen de symbolen bij deze toetsen hebben gevormd naar de letters van het alfabet. Hoe we die vorm kunnen veranderen gaan we nu bespreken!
Allereerst moeten we gaan bepalen, hoe we de vorm van een symbool kunnen vastleggen met behulp van inkt- en papierpunten. Dat kan, bijvoorbeeld door een papierpunt voor te stellen door een "0" en een inktpunt door een "1". Op die manier zou u de vorm van een hondje kunnen opbouwen uit onderstaand patroon van nullen en enen:

```
1 1 0 0 0 0 0 1
1 1 0 0 0 0 1 0
0 0 1 1 1 1 1 0
0 0 1 1 1 1 1 0
0 0 1 1 0 1 1 0
0 1 1 0 0 0 1 1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

Een symbool van de Spectrum wordt immers opgebouwd uit een een patroon van 8 bij 8 punten. Ziet u niet dadelijk het beeld van een hondje in de combinatie van "1" en "0", probeer dan eens alle "1"-en zwart te kleuren. Nu we begrijpen hoe we dit symbool kunnen opbouwen uit een kombinatie van "1"-en en "0"-en, kunnen we gemakkelijk elk willekeurig symbool samenstellen. Het volgende probleem is hoe we die kombinatie van binaire gegevens in de computer kunnen voeren.
Zoals reeds opgemerkt, kunnen we de "U-D"-symbolen onderbrengen bij de A- tot en met U-toetsen van de computer.
De specifieke samenstelling van het 8 bij 8 patroon voor een bepaald symbool wordt in het geheugen van de Spectrum regel voor regel opgeslagen.
Om de geheugeninhoud van een regel te veranderen, moeten we een tamelijk gecompliceerde boodschap aan de computer doorgeven:

```
POKE USR "b"+2,BIN xxxxxxxx
```

Uiteraard wordt in de praktijk de ''xxxxxxxxx'' vervangen door de combinatie van ''1''-en voor die regel. Het is niet noodzakelijk precies te begrijpen wat we de computer in het oor fluisteren om op deze manier eigen symbolen in het geheugen te programmeren. We moeten alleen rekening houden met het feit dat de regels zijn genummerd van 0 tot en met 7. In het bovenstaande voorbeeld wordt dus de derde regel gewijzigd door het intoetsen van ''b + 2'' na de ''USR''-instructie.
De algemene definitie voor het veranderen van de nde regel van het symbool C is:

```
POKE USR "C"+n-1,BIN xxxxxxxx
```

We moeten natuurlijk alle 8 regels op deze manier invoeren, willen we een kompleet nieuw symbool in het geheugen van de computer zetten.
Zo zal het ''hondjes''-symbool onder de toets van de letter A worden opgenomen, als we het volgende programma in de computer invoeren:

```
10 POKE USR "a"+0,BIN 11000001
20 POKE USR "a"+1,BIN 11000010
30 POKE USR "a"+2,BIN 00111110
40 POKE USR "a"+3,BIN 00111110
50 POKE USR "a"+4,BIN 00110110
60 POKE USR "a"+5,BIN 01100011
70 POKE USR "a"+6,BIN 00000000
80 POKE USR "a"+7,BIN 00000000
90 PRINT "[A]"
```

Dit programma definiëert het nieuwe symbool in de regels 10 tot en met 80 en print het symbool op het scherm door de ''PRINT''-instructie van regel 90.
Het woord is nu aan de kunstenaar! Alleen onze verbeelding beperkt de vormenrijkdom aan ruimteschepen, raketten, schepen, auto's, en zo verder, die we met de ''U-D''-symbolen op het scherm kunnen tekenen!
In dit boek zullen we steeds opnieuw teruggrijpen naar de mogelijkheden van de ''U-D''-symbolen.

Zoals reeds gezegd, is het niet noodzakelijk om exact te weten wat we doen. U kunt deze paragraaf dus rustig overslaan. Het is natuurlijk wél interessant. Vandaar dat we zullen proberen u uit te leggen wat de computer met de ''USR''-functie doet. De functie ''USR-letter'' geeft ons, de gebruiker (user), toegang tot die geheugenlocatie waarin de eerste regel van het bij de letter behorende symbool is ondergebracht.
Vandaar is:

```
USR "letter"+1
```

de geheugenlocatie (address) van de tweede regel van de ''letter''.
De functie ''BIN'' vertaalt een regel enen en nullen naar een getal. Dat volgt uit het volgend programmaatje:

```
10 PRINT BIN 011
```

waarin we ''0''-en en ''1''-en kunnen opnemen. Die vertaling kan plaatsvinden, doordat de computer een regel ''0''-en en ''1''-en interpreteert als een binair getal (vandaar de naam van de instruktie). Dit binaire getal wordt dan in de geselecteerde geheugenplaats opgeslagen door de ''POKE''-instructie. Dit wordt in hoofdstuk 7 uitvoerig behandeld. Als het geheugen op het scherm wordt uitgelezen, resulteert het op de geheugenlocatie opgenomen binaire getal in het juiste patroon van inkt- en papierpunten.

**Het gebruik van kleur**

Het basis-idee voor het programmeren van kleur op de Spectrum hebben we reeds besproken: de ''INK''- en ''PAPER''instrukties. Er zijn echter een aantal begrippen - nodig voor het juiste gebruik van deze instructies - die nog niet zijn behandeld. Het eerste probleem is dat een ''INK''- of ''PAPER''instructie de kleur van ALLE punten bepaalt, die op de instructie volgen. Deze instructies beïnvloeden dus niet de kleur van alles wat al op het scherm staat. Probeer het volgende programma even uit:

```
10 INPUT "Paper kleurnummer ? ";p
20 INPUT "Ink kleurnummer ? ";i
30 PAPER p
40 INK i
50 FOR j=1 TO 32
60 PRINT "X";
70 NEXT j
80 GOTO 10
```

Nadat we in dit programma de gewenste inkt- en papierkleuren hebben ingevoerd, wordt een lijn volgeschreven met x-en in de gekozen kleur. Hoewel iedere nieuwe regel in een andere kleur wordt geschreven, blijven de kleuren van de reeds geschreven lijnen onveranderd. Bij ge-

bruik van een zwart-wit toestel zien we de kleuren als verschillende tinten grijs. Met dit programma kunnen we ook het effekt van gelijke inkt- en papierkleuren uitproberen. Hoewel we nu geen X-en meer op het scherm zien, moeten we ons wel realiseren dat ze er wel zijn!
Hoewel deze manier van kleurcontrole zeer eenvoudig is, is hij er onhandig als we in een regel diverse kleuren willen opnemen. Toch kunnen we dat doen, als we de "INK"- en "PAPER"-instructies opnemen in een "PRINT"-regel. Het volgende programma

```
10 PRINT INK 0;PAPER 7;"X";INK 7;PAPER 0;"X"
```

schrijft eerst een zwarte X op een witte achtergrond en nadien een witte X op een zwarte achtergrond. Wat we moeten onthouden is dat het effect van "INK"- en "PAPER"-instructies in een "PRINT"-regel beperkt blijft tot de regel zelf. Na de "PRINT"-instructie worden de kleuren weer wat ze waren voor de "PRINT"-instructie. Vervolgens gaan we onze aandacht richten op twee nieuwe instructies, die van belang zijn bij het gebruik van kleur: "CLS" en "BORDER".
De "CLS"-instructie veegt het scherm schoon (clear), althans dat is wat we zien. In werkelijkheid veranderdert deze instructie alle punten op het scherm in papierpunten. Het volledige scherm krijgt dus de kleur die we door middel van een "PAPER"-instructie hadden vastgelegd.
De "BORDER"-instructie bepaalt de kleur van de rand rond dát deel van het scherm, waarin de Spectrum kan schrijven. Het enige wat we verder van deze instructie moeten weten, is dat hij ook de kleur bepaalt van het gebied waarin we op een bepaald moment werken.
Een voorbeeld met "CLS"- en "BORDER"-instructies is:

```
10 FOR i=0 TO 7
20 PAPER i
30 CLS
40 BORDER 7-i
50 PAUSE 5
60 NEXT i
```

Indien u de kleuren iets trager wilt laten variëren, is het voldoende het cijfer achter de "PAUSE"-instructie te verhogen.
Als laatste voorbeeld van het gebruik van kleuren geven we onderstaand programma, waarmee we een aantal gekleurde balken op het scherm kunnen laten verschijnen.

```
10 FOR c=0 TO 7
20 PAPER c
30 PRINT " ";:REM 4 spaties
40 NEXT c
50 GOTO 10
```

**De "INVERSE"-en "OVER"-instructies**
De twee genoemde instructies hebben als bijzonderheid dat ze de manier waarop symbolen op het scherm worden geschreven beïnvloeden. Het bevel "INVERSE 1" heeft tot gevolg dat alle daaropvolgende symbolen geïnverteerd op het scherm verschijnen: dus inktpunten nemen de plaats in van papierpunten en vice versa. Willen we terug naar de normale manier van schrijven, dan moeten we het bevel "INVERSE 0" geven. Denk er aan, dat het "INVERSE"-bevel géén nieuwe symbolen tot gevolg heeft. Het verwisselt alleen de inkt- en papierpunten bij het schrijven van een symbool op het scherm.
Het resultaat van een "INVERSE"-instructie volgt uit onderstaand programma:

```
10 INVERSE 1
20 INPUT a$
30 PRINT a$
40 GOTO 20
```

Nadat dit programma door de machine is uitgevoerd, moeten we een "INVERSE 0"-bevel geven, waarna de computer weer in zijn normale werking wordt geschakeld.
Het "OVER 1"-bevel beïnvloedt ook de manier waarop inkt- en papierpunten op het scherm worden geschreven. Alleen is dit niet zo simpel uit te leggen als de "INVERSE"-instructie. Wat er na een "OVER 1"-instructie op het scherm gebeurt bij het schrijven van een nieuw symbool is namelijk afhankelijk van wat er op die plaats reeds op het scherm te zien was. Als algemene regel geldt dat als een punt van het nieuwe symbool samenvalt met een punt van het oude symbool er op die plaats een papierpunt ontstaat. Bij niet samenvallende punten ontstaan inktpunten. Eenvoudig gezegd komt het hierop neer, dat als u een nieuw symbool intikt op de plaats van een oud, beide symbolen op het scherm verschijnen. Daar waar ze echter samenvallen, verschijnt een papierpunt.

Een voorbeeld:

```
10 CLS
20 OVER 1
30 PRINT AT 0,0;"A"
40 PRINT AT 0,0;"_"
50 PRINT AT 1,0;"A"
60 PRINT AT 1,0;"\"
70 OVER 0
```

Het eerste "PRINT"-bevel plaatst de letter A in de bovenste linkerhoek van het scherm. De tweede "PRINT"-instructie zet een streepje op dezelfde plaats onder de letter. We hebben immers op regel 20 een "OVER 1"-bevel gegeven! Door de regels 50 en 60 zet de computer de letter A en het symbool op dezelfde plaats op het scherm. Als we echter goed kijken, dan zien we dat beide symbolen wegvallen waar ze elkaar overlappen.
We kunnen de "OVER"-instructie ook gebruiken voor het printen en nadien wissen van een symbool. U kunt bijvoorbeeld de woorden "Een bericht" laten printen en nadien laten verdwijnen door een tweede "PRINT"-bevel na een "OVER 1"-instructie op te nemen.
We tikken het volgende programma in

```
10 CLS
20 PRINT AT 0,0;"Een bericht"
30 OVER 1
40 GOTO 20
```

en zien eerst de woorden "EEN BERICHT" oplichten en nadien verdwijnen.
De gevolgen van "OVER 1"- en "INVERSE 1"-bevelen blijven gelden tot men ofwel "OVER 0" en "INVERSE 0" invoert, ofwel de machine uitschakelt. Net zoals "INK" en "PAPER" kunnen ook deze instructies in een "PRINT"-bevel worden toegepast. Hun effect is dan ook plaatselijk, namelijk begrensd tot de lengte van het "PRINT"-bevel.
De instructies "INVERSE" en "OVER" zullen uitvoerig aan de orde komen als we het gaan hebben over het programmeren van grafieken met hoge resolutie.

**De "BRIGHT"- en "FLASH"-instructies**

De twee laatste instructies die bijdragen aan het schrijven van symbolen op het scherm hebben, net zoals de "INK"- en "PAPER"-bevelen, geen invloed op het patroon van inkt- en papierpunten.
Wél beïnvloeden zij de manier waarop de inkt- en papierpunten op het scherm verschijnen.

Na een "BRIGHT 1"-bevel worden alle inkt- en papierpunten met een verhoogde helderheid weergegeven. Na een "FLASH 1"-instructie zullen alle punten gaan knipperen, dat wil zeggen voortdurend omschakelen van papier- naar inktkleur en vice versa.
Ook deze twee instructies kunnen alleen maar worden opgeheven door de "BRIGHT 0"- en "FLASH 0"-bevelen in te toetsen.
Het volgende programma verduidelijkt de werking van deze bevelen:

```
10 CLS
20 PAPER 1
30 INK 7
40 PRINT "Dit is gewoon"
50 BRIGHT 1
60 PRINT "Dit is helder"
70 BRIGHT 0
80 FLASH 1
90 PRINT "Dit is knipperend"
100 BRIGHT 1
110 PRINT "Dit is helder en knipperend"
120 BRIGHT 0
130 FLASH 0
```

Men kan "BRIGHT" en "FLASH" samen gebruiken, waardoor helder knipperende mededelingen ontstaan.
Hoe een symbool er uit ziet, wordt bepaalt door het patroon van inkt- en papierpunten en de manier waarop ieder individueel punt van dat patroon op het scherm verschijnt. De enige manier om het patroon te variëren, is gebruik te maken van de "INVERSE" en "OVER"-functies en het invoeren van eigen "U-D"-symbolen.
De "INK"-, "PAPER"-, "FLASH" en "BRIGHT"-functies bepalen de manier waarop individuele punten op het scherm verschijnen.
Dit geeft eigenlijk een goed inzicht in de manier waarop de Spectrum omgaat met zijn TV-scherm. Een gebied van het geheugen wordt gebruikt voor het opslaan van patronen van inkt- en papierpunten en een ander gebied bevat de informatie over hoe ieder individueel punt moet worden weergegeven. De gegevens, die bepalen hoe een punt er zal uitzien op het scherm, worden de "attributen" van dat punt genoemd en bijgevolg noemt men dat tweede geheugendeel ook wel het "attributendeel".

Over deze nu nog vreemde benaming "attributen" zullen we in hoofdstuk 10 veel meer vertellen.

**Het "attribuut 8" en de "kleur 9"**

Vaak komt het in de praktijk voor dat we iets op een bepaalde plaats van het scherm willen schrijven zonder de "attributen" van die "symboolplaats" verloren te laten gaan. Als u nog weet wat die attributen precies waren, is er geen probleem. U moet dan alleen maar de juiste attributen-instructies intoetsen. Als u wist dat een bepaalde lokatie knipperde, dan moet u het bevel "FLASH 1" geven en alles is in orde.

Maar als u niet meer precies weet hoe de attributen-instructies voor een bepaalde symboolplaats luidden, dan zit u in moeilijkheden, tenzij u op de hoogte bent van het begrip "attribuut 8".

Het toevoegen van het cijfer 8 aan een van de attributen-instructies "BRIGHT", "FLASH", "INK" en "PAPER" heeft tot gevolg dat ieder nieuw symbool op het scherm dezelfde attributen krijgt als het symbool op die plaats had.

Na een "FLASH 8"-bevel zal ieder symbool gaan knipperen als het op een symboollokatie terecht komt, die al een "FLASH" als attribuut had. Op de andere lokaties zullen de nieuwe gegevens gewoon, dus niet knipperend worden weergegeven.

Na een "PAPER 8"-instructie zullen alle symbolen dezelfde kleur krijgen als de symbolen die voorheen op deze lokaties stonden. Vanwege de bijzondere eigenschappen van "attribuut 8" noemt men een bevel met een 8 ook wel eens een "transparant" bevel, omdat het de originele attributen van een symboollokatie laat doorschemeren.

Een gelijksoortige omstandigheid kan zich voordoen met betrekking tot de papier- en inktkleuren.

Wat voor soort kleur moet u programmeren om er absoluut zeker van te zijn dat wat u intikt ook duidelijk zichtbaar is tegen de kleur van de achtergrond?

U hoeft het natuurlijk niet in uw hoofd te halen een tekst zwart te laten afdrukken, als de kleur van de papierpunten voorheen als zwart was ingeprogrammeerd!

Ook dit is gemakkelijk op te lossen, als u nog weet welke papierkleur is gekozen. U kiest dan voor de inktpunten een contrasterende kleur. Maar als u niet meer precies weet hoe de achtergrondkleur in het vorige programma was, of als de kleur van de achtergrond vaak wijzigt, dan zit u weer in de problemen.

Maar ook nu is er een simpele oplossing. U kunt de "verkeerde" kleurencode 9 gebruiken. Dan weet de computer dat hij een kleur moet selecteren, die sterk contrasteert met de kleur die op een bepaalde lokatie als achtergrond is gebruikt. Zo zal bijvoorbeeld na een "INK 9"-code de tekst zwart op het scherm verschijnen, als de achtergrondkleur licht is, dus code 4 tot en met 7 heeft. Anderzijds, als de achtergrond donker is (papiercode 0 tot en met 3), dan zal de tekst wit worden geschreven.

Hetzelfde verhaal geldt voor de "PAPER 9"-instructie. Hiermee kiest de computer de kleur van het papier contrasterend met de bestaande inktkleur. "Kleur 9" zorgt er dus voor dat wat op het scherm verschijnt goed leesbaar zal zijn. Maar bedenk wel dat met "kleur 9" de computer of zwart of wit kiest, maar nooit een kleur. U krijgt dus wel tamelijk saaie prentjes op het scherm.

**Conclusie**

In dit hoofdstuk hebben we ons bezig gehouden met de fundamentele systemen waarmee de Spectrum de inhoud van een schermbeeld kan vaststellen. We zijn er ons van bewust, dat we nog geen indrukwekkendde staaltjes van mooie grafische voorstellingen hebben laten zien.

Maar het is absoluut onmogelijk verder te gaan zonder eerst de technieken voor het opbouwen van prentjes met lage resolutie onder de knie te hebben.

Deze technieken zullen in het vervolg steeds opnieuw opduiken en worden verfijnd.

# 3. SPELEN MET TOEVAL

In het abstracte denken is "toeval" een tamelijk omstreden begrip en men kan zich afvragen wat een dergelijk filosofisch geladen woord moet in een boek over het programmeren van spelletjes op een computer. Het valt echter niet te ontkennen dat toeval, of geluk, een zeer wezenlijk bestanddeel vormt van bijna alle soorten spelletjes.

Er zijn zuivere geluksspelletjes: spelletjes met kaarten of met dobbelstenen. Daarnaast zijn er spelletjes waarbij de snelheid waarmee u reageert op een onverwachte gebeurtenis van belang is. Tot slot zijn er spelletjes waarbij u uw verstand moet gebruiken om een tegenspeler te verslaan wiens beslissingen u niet kunt voorzien.
Toevalsgetallen vormen het hart van dergelijke spelletjes.

**Pseudo-toeval!**
Wat is de definitie van een willekeurig of toevalsgetal? We hebben het antwoord al min of meer gegeven. Het is een getal dat met geen mogelijkheid is te voorspellen.
Het resultaat van "kruis of munt" is zuiver toeval, net zoals het resultaat van een dobbelsteenworp. Tenzij u helderziende bent, kunt u het resultaat van het opgooien van een muntstuk niet voorspellen. Net zo min als u van tevoren kunt zeggen welke zijde van een dobbelsteen boven zal blijven liggen.
Het feit dat de spelers het resultaat niet kunnen beïnvloeden is het belangrijkste aspect van het gebruik van toevalsgetallen bij spelletjes.
Misschien heeft u zich afgevraagd of het voor een computer niet onmogelijk is willekeurige getallen op te wekken.
Zo'n machine kan immers alleen maar het resultaat berekenen van een formule die er eerst is ingetoetst. Dit is inderdaad juist. Een machine kan alleen maar getallen produceren, die het resultaat zijn van een berekening. En het resultaat van een berekening kan nooit onvoorspelbaar zijn en dit is immers de voornaamste eigenschap van de echte willekeur.

Wat we een computer wél kunnen laten berekenen, is een reeks getallen waarvan ieder volgend getal zeer moeilijk te voorspellen is. Zo'n reeks getallen noemen we pseudo-willekeurig (pseudo-random).
We moeten zeer duidelijk voor ogen houden dat een pseudo-willekeurig getal nooit het resultaat kan zijn van een willekeurig getal nooit het resultaat kan zijn van een willekeurig proces als het werpen van een dobbelsteen. Het is dus in theorie voorspelbaar, maar de wetten van die voorspelbaarheid zijn zo ingewikkeld dat een buitenstaander nooit kan berekenen wat het volgende getal zal zijn.
We kunnen dus stellen, dat een computer onvoorspelbare getallen kan genereren, in plaats van willekeurige getallen. Een computer gebruikt echter de steeds een formule, ook voor het berekenen van deze onvoorspelbare getallen. Iedereen, die een kopie van de toegepaste formules bezit, kan dus in theorie ieder volgend nummer uit de reeks berekenen. In de praktijk zijn de formules echter zo ingewikkeld, dat alleen een wiskundig genie in staat is om binnen de beschikbare tijd de formules uit te rekenen.

**De "RND"- en "RAND"-instructies**
Met de "RND"-functie kunnen we de Spectrum pseudo-willekeurige getallen laten opwekken tussen 0 en 1. Iedere keer dat de Spectrum de "RND"-instructie tegenkomt, berekent hij het volgende getal uit de reeks. De "RND"-functie heeft een belangrijke eigenschap: ieder getal tussen 0 en 1 heeft even veel kans om op het scherm te verschijnen!
Een meer wiskundige manier om hetzelfde te zeggen is: de "RND"-functie produceert "gelijkmatig verdeelde" pseudo-toevals getallen tussen 0 en 1.
Nu u weet wat de "RND"-functie doet, kunnen we ons bezighouden met het praktische gebruik ervan. Een demonstratie van deze functie krijgt u als u het volgende programma invoert:

```
10 PRINT RND
20 GOTO 10
RUN
```

Het scherm wordt volgeschreven met getallen tussen 0 en 1, zoals bijvoorbeeld:

```
.0011291504
.08581543
```

Valt u nu niet een bepaalde samenloop van omstandigheden op? Hebben we de twee eerste getallen bovenaan het scherm juist voorspeldt? Is dat niet het geval, schakel uw computer dan even uit en weer aan en probeer opnieuw. Uw scherm zal nu wél een reeks getallen tonen, waarvan de twee eerste gelijk zijn aan de twee bovenstaande.
Dat valt te verklaren uit het feit dat de pseudo-toevalsgetallen bij alle machines worden opgewekt volgens één en dezelfde formule, een formule die u bovendien in de handleiding van de spectrum kunt vinden. Zo geheimzinnig is het dus niet!
Als u dit programma diverse malen door de machine laat uitvoeren, zult u zien dat de reeks getallen gewoon doorloopt.
Het feit dat de reeks volledig herhaalbaar is, is voor sommige toepassingen zeer praktisch, bijvoorbeeld wanneer u een bepaald patroon van toevalsgebeurtenissen wilt herhalen. Voor andere toepassingen is deze eigenschap echter volkomen onbruikbaar. Het spelen van kansspelletjes op de Spectrum zou heel erg snel gaan vervelen.

Gelukkig hebben we de beschikking over de "RAND"-functie. In werkelijkheid print de machine het woord "RANDOMIZE" op het scherm als we de "RAND"-toets bedienen. Om de programma's wat eenvoudiger in te tikken, zullen we echter het woord RAND blijven gebruiken.
De "RAND"-instructie bepaalt op welk punt van de reeks de computer start met het schrijven van de getallen. Dit kan zowel een vastgesteld als een willekeurig punt zijn. Het programma, waarmee we het startpunt bepalen, luidt: "RAND " gevolgd door een willekeurig getal. Probeer eens:

```
10 RAND 35
20 PRINT RND
30 GOTO 20
```

Laat dit programma een paar keer uitvoeren. De volgorde der getallen is steeds gelijk.

Als u het volgende programma laat uitwerken:

```
10 RAND 35
20 PRINT RND
30 GOTO 10
```

zult u constateren dat de computer steeds hetzelfde nummer uitschrijft: het startpunt van de getallenreeks, bepaald door "RAND 35"!
Als u telkens een ander gedeelte van de reeks op het scherm wilt zien verschijnen, moet u het volgende programma invoeren:

```
10 RAND 0
```

Bij de "RAND 0"-instructie bepaalt de computer zélf het startpunt van de reeks en wel door middel van de stand van een inwendige teller, die het aantal televisiebeelden telt dat de machine op het scherm van uw TV heeft geschreven sinds de computer is ingeschakeld. In hoofdstuk 8 gaan we dieper op dit telsysteem in.
Er bestaat dus een wiskundig verband tussen het startpunt van de reeks en de tijd dat uw computer aan staat. Door de snelheid van de teller, 50 beelden per seconde, is het in de praktijk onmogelijk om te voorspellen wat de startpositie zal zijn.
Laten we even samenvatten wat we te weten zijn gekomen over de pseudo-toevalsmogelijkheden van de machine.
De "RND"-functie berekent het eerstvolgende getal uit de getallenreeks en met de "RAND"-functie kunnen we de startpositie in die reeks bepalen.
Als bewijs het volgende programma:

```
10 RAND 0
20 PRINT RND
30 GOTO 10
```

Dit programma schrijft een aantal startpunten op het scherm, die worden bepaald door de "RAND 0"-instructie. Hoewel de startwaarde toeneemt met de tijd, is de exacte waarde niet te voorspellen.
Vandaar dat de meest willekeurige reeks getallen die we met de computer kunnen opwekken op het scherm verschijnt, als het het volgende programma intoetsen:

```
10 RAND 0
20 PRINT RND
30 GOTO 20
```

**En nu de praktijk!**

Op de keeper beschouwd kunnen we niet zo erg veel doen met de reeks getallen die op het scherm verschijnt na het intoetsen van het vorige programma. Laten we dus maar eens aan de hand van een eenvoudig voorbeeld bekijken hoe we de theorie kunnen omzetten in iets praktisch bruikbaars. Als voorbeeld kiezen we het opgooien van een munt. Er zijn slechts twee mogelijkheden; ofwel ''munt''. Op de een of andere manier moeten we de tamelijk groffe resultaten van de ''RND''-functie zo vereenvoudigen dat de computer of met ''kruis'' of met ''munt'' antwoordt en wel zo, dat beide mogelijkheden even veel kansen krijgen.

Welnu, dat gaat als volgt: we splitsen de getallenreeks in twee even grote delen. Zoals we weten, genereert de ''RND''-functie getallen tussen 0 en 1. Het midden daarvan is 0,5:

```
I----------I----------I
0          .5         1
```

Omdat de ''RND''-functie gelijkmatig verdeelde getallen oplevert tussen 0 en 1, zullen er even veel getallen ontstaan die kleiner zijn dan 0,5 als getallen die groter zijn dan die waarde. Het is voldoende alle getallen die kleiner zijn dan 0,5 ''kruis'' te noemen en alle getallen die groter zijn dan 0,5 ''munt''.

Het volgende programma geeft de computer die opdracht:

```
10 RAND 0
20 LET r=RND
30 IF r<.5 THEN PRINT "kruis"
40 IF r>.5 THEN PRINT "munt"
50 GOTO 20
```

Met regel 10 zorgen we voor een willekeurig startpunt en met regel 20 maken we ''r'' gelijk aan het berekende willekeurige getal. De regels 30 en 40 zorgen ervoor dat ofwel ''kruis'' ofwel ''munt'' op het scherm verschijnt. Zo eenvoudig gaat dat! Natuurlijk hebben we met dit programma nog geen erg opwindend spel ontwikkeld, maar komen we in een later stadium op deze toepassing terug. Kunnen we ook een programma schrijven voor die mensen, die met een ''vervalste'' munt willen spelen? Dat gaat net zo eenvoudig. We moeten dan de getallenreeks verdelen in twee ongelijke delen. Als we de kans op ''kruis'' gelijk stellen aan ''p'', dan luidt het algemene programma:

```
10 RND 0
20 INPUT p
30 LET r=RND
40 IF r<p THEN PRINT "kruis"
50 IF r>=p THEN PRINT "munt"
60 GOTO 30
```

Voor het simuleren van het opgooien van een munt hebben we maar met twee mogelijkheden rekening te houden, al dan niet met evenveel kansen. Natuurlijk zijn er situaties denkbaar, waar we met meer dan twee mogelijke resultaten te maken krijgen en waar niet ieder antwoord dezelfde kans heeft. Zo lijkt het weer in ons landje voor de gewone man een typisch verschijnsel dat meer met toeval dan met wetenschappelijke studie te maken heeft. We zouden dus een programma kunnen verzinnen, waarmee we kunnen controleren of de geleerden inderdaad het weer voorspellen volgens wetenschappelijke normen of eerder een beroep doen op hun ervaring. Dit programma combineert toevals-elementen met onze kennis van het gemiddelde weerpatroon in een bepaald seizoen. Uit onze ervaring weten we dat we gedurende de 100 dagen die een lente duurt ongeveer 40 zonnige 30 bewolkte, 20 regenachtige en 10 dagen met sneeuw kunnen verwachten. Statistisch kunnen we dat als volgt uitdrukken: 0,40 zonnige dagen, 0,20 regenachtige dagen, 0,30 bewolkte dagen en 0,10 dagen met sneeuw. Op een lijn, lopende van 0 tot en met 1, kunnen we vier stukken afmeten, waarvan de lengte overeenkomt met de kans op ieder van de vier weertypen.

```
   zonning      bewolkt      regen  sneeuw
     .4           .3           .2     .1

I----------I-------I-----I--I
0          .4      .7     .9 1
```

Als we willekeurige getallen door de computer laten berekenen, is de kans dat een getal in een van de vier secties van de lijn valt, evenredig met de lengte van de vier delen. Op die manier kunnen we het weer met een tamelijk grote nauwkeurigheid voorspellen.

Het weertype dat overeenkomt met de lijnsectie waarin een door de computer berekend wille-

keurig getal valt, is het type dat wordt voorspeld.
Zo zal de computer "bewolkt" voorspellen als hij het getal 0,6712348117 heeft berekend.
Toch heeft deze eenvoudige methode een onduidelijkheid. Welk type weer wordt er voorspeld als de computer een getal berekend dat precies samenvalt met een van de scheidingspunten tussen de diverse secties, zoals bijvoorbeeld het getal 0,4?
Moet de voorspelling dan "zonnig" of "bewolkt" luiden? In feite maakt het niet zoveel uit wat we kiezen, als we maar een bepaalde systematiek handhaven. Zo kunnen we bijvoorbeeld het cijfer 0 bij het zonnige gebied rekenen en ieder volgend grensgetal bij het daaropvolgend weertype. Maar dan blijven we wél mooi zitten met het getal 1!
Dat is geen probleem, want als we ons de definitie van de "RND"-functie voor de geest halen, zullen we ons ongetwijfeld herinneren, dat deze functies getallen berekent van 0 tot 1, maar niet tot en met 1, want dat komt nooit voor. Het onderstaande gegevens programma voor het voorspellen van lenteweer zal wel geen nadere toelichting behoeven.

```
10 RAND 0
20 LET r=RND
30 PRINT "Weerbericht"
40 IF r<.4 THEN PRINT "Zonnig"
50 IF r>=.4 AND r<.7 THEN PRINT "Bewolkt"
60 IF r>=.7 AND r<.9 THEN PRINT "Regen"
70 IF r>=.9 THEN PRINT "Sneeuw"
```

Het enige moeilijke punt van dit programma is het precies bepalen van de secties, waarin de toevalsgetallen vallen. Hiervoor verwijzen we naar het lijnpatroon, wat dit verduidelijkt.
We kunnen talloze truukjes toepassen, waardoor dit programma óf minder geheugen vergt, óf sneller door de machine wordt verwerkt. Het bovenstaande programma is echter het gemakkelijkst te begrijpen en kan bovendien op iedere computer worden toegepast.

**Willekeurige gehele getallen**

Een manier om te bepalen welke gebeurtenis zal plaatsvinden, is het verdelen van het interval tussen de getallen 0 en 1. Maar dat is niet de enige manier. Als we te maken hebben met een bepaald aantal gebeurtenissen die ieder even veel kans maken, bestaat er een alternatief: het vermenivuldigen van het door de computer berekende getal met het aantal gebeurtenissen, die resultaat afronden naar een geheel getal en hierbij het cijfer 1 optellen.
Dit klinkt allemaal erg ingewikkeld, maar dat is het niet! Laten we als voorbeeld het gooien van een dobbelsteen behandelen. We kunnen daarbij kiezen uit zes mogelijkheden. We zouden natuurlijk de reeds beschreven methode kunnen gebruiken door de 0 tot 1-lijn in zes gelijke delen te verdelen, maar laten we het eens volgens de nieuwe formule proberen. Als we het eindresultaat van "RND" met zes vermenivuldigingen, ontstaan getallen tussen 0 en 6. Als we nadien de "INT"-instructie gebruiken, wordt het resultaat omgezet in een geheel getal tussen 0 en 5. Het cijfer 1 daarbij opgeteld geeft een cijfer tussen 1 en 6 als uiteindelijk resultaat.
Met het volgende programma kunnen we ons vertrouwd maken met dit systeem:

```
10 RAND 0
20 LET r=RND
30 PRINT r
40 LET r=r*6
50 PRINT r
60 LET r=INT(r)
70 PRINT r
80 LET r=r+1
90 PRINT r
```

Als u dit programma een aantal malen laat uitvoeren, wordt snel duidelijk, wat er precies gebeurt.
Natuurlijk is dit een zuiver educatief programma: in de praktijk kunnen we door middel van slechts één instructie hetzelfde doel bereiken:

```
10 RAND 0
20 LET r=INT(RND*6)+1
30 PRINT r
40 GOTO 20
```

De algemene formule voor het berekenen van willekeurige gehele getallen tussen n en m luidt:

```
10 LET r=INT(RND*(m-n+1))+n
```

Meestal kunnen we n gelijk stellen aan 1, waardoor de formule zich vereenvoudigt tot:

```
10 LET r=INT(RND*m)+1
```

Als we m gelijk stellen aan 6, vinden we het dobbelsteenprogramma terug.

Wel moeten we ons goed realiseren dat dit zeer eenvoudige systeem alleen bruikbaar is als alle gebeurtenissen even vaak mogen voorkomen.

**Twee verbeterde programma's**

Ondanks het feit dat we vrij diep zijn doorgedrongen in het verschijnsel "willekeur" hebben we nog geen echt volledige spelletjesprogramma's ontwikkeld. Meestal vormen de behandelde programma's slechts een deel van een veel uitgebreider geheel. Toch kunnen we ook de twee kleine eerder behandelde programma's verbeteren.

We gaan ons eerst bezig houden met het programma voor het opgooien van een munt.

Computer-programma's missen vaak de hoog nodige spanning. Een munt wordt opgegooid . . . hij cirkelt door de lucht . . . valt en wankelt heen en weer . . . wordt het munt? . . . of kruis? . . . eindelijk ligt hij stil!

Een computer-programma, daarentegen, schrijft gewoon het woordje "kruis" of het woordje "muntje" op het scherm en dat nog wel in een oogwenk! Vandaar dat we gaan proberen het programma te vertragen, waardoor er wat meer spanning ontstaat.

Toets het volgende programma in:

```
10 DIM b$(2,5)
20 RAND 0
30 INPUT "Wilt u speten j/n ?";a$
40 IF a$<>"j" THEN STOP
50 INK 0:PAPER 6:CLS
60 INPUT "kruis of munt ? ";a$
70 PRINT "U heeft dus gekozen voor ";a$
80 LET r=INT(RND*15)+10
90 LET b$(1)="kruis"
100 LET b$(2)="munt"
110 LET k=0
120 FOR i=1 TO r
130 LET k=NOT k
140 PRINT AT 5,0;b$(k+1)
150 FOR j=1 TO 1
160 NEXT j
170 NEXT i
180 IF a$(1)=b$(k+1,1) THEN PRINT "U wint"
190 IF a$(1)<>b$(k+1,1) THEN PRINT "U verliest !!"
200 GOTO 30
```

Dit programma werkt volgens een ander principe dan het vorige. Met de regels 10, 90 en 100 bouwen we een "string" op rond de woorden "kruis" en "munt". De "FOR"-lus (loop) van regel 120 tot en met regel 170 zal er voor zorgen dat een van beide woorden wordt afgedrukt. Regel 130 brengt u mischien in verwarring. "NOT k" kent aan k de waarde 0 toe als die 1 was, en omgekeerd. Dit door de lus veroorzaakte "knipperen" van k zorgt voor het beurtelings afdrukken op het scherm van "kruis" of "munt". Als k = 0 zal door regel 140 het woord "kruis" worden geschreven, k = 1 daarentegen zorgt voor het begrip "munt".

Het toevalselement wordt in regel 80 geïntroduceerd, door met behulp van r te bepalen hoe vaak de lus wordt doorlopen. Het is zonder meer duidelijk, dat een oneven waarde van r het woord "kruis" doet verschijnen en bij een even waarde het woord "munt" verschijnt.

De instructies in de regels 150 en 160 introduceren een vertraging, waardoor de twee woorden elkaar in een steeds trager tempo afwisselen, wat het spannender maakt.

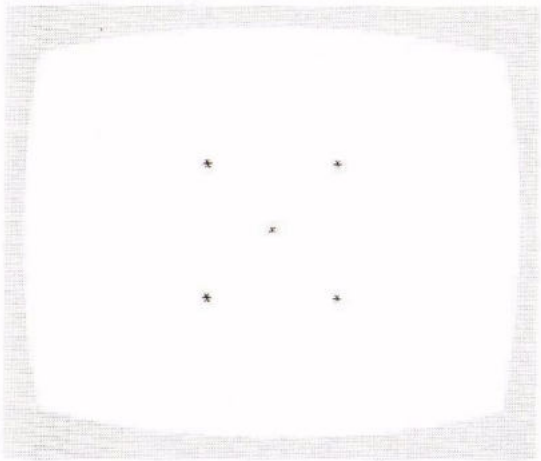
Toegegeven, dit programma is alles behalve eenvoudig! Als u het niet allemaal kan volgen, moet u niet in paniek geraken. Diverse technieken die we in dit programma hebben gebruikt, zullen in de rest van dit boek uitvoerig aan de orde komen.

Als tweede te verbeteren programma komt natuurlijk het dobbelsteenspel in aanmerking.

Het ligt voor de hand waaruit deze verbetering zal bestaan: we zullen de computer zo programmeren, dat het bekende patroon van zwarte puntjes op het scherm verschijnt. Even nadenken kan ons veel werk besparen. Zo is het duidelijk dat het patroon voor worp 3 gelijk is aan de som van de patronen voor worp 1 en worp 2. Patroon 4 ontstaat door aan het patroon van worp 2 twee extra punten toe te voegen. Hetzelfde geldt voor patroon 5 (4 + 1) en patroon 6 (patroon 4 + 2 extra punten).

Het programma ziet er als volgt uit:

```
10 RAND 0
20 LET r=INT(RND*6)+1
30 GOSUB r*100
40 INPUT a$
50 IF a$="s" THEN STOP
60 INK 2:PAPER 6:CLS
70 GOTO 20
100 PRINT AT 5,5;"*"
110 RETURN
200 PRINT AT 0,0;"*"
210 PRINT AT 10,10;"*"
220 RETURN
300 GOSUB 100
310 GOTO 200
400 PRINT AT 0,10;"*"
410 PRINT AT 10,0;"*"
420 GOTO 200
500 GOSUB 400
510 GOTO 100
600 PRINT AT 5,0;"*"
610 PRINT AT 5,10;"*"
620 GOTO 400
```

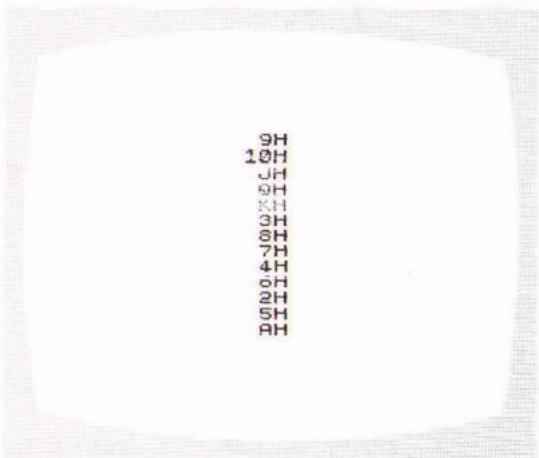


Het enige opmerkenswaardige aan dit programma is regel 30, waar aan de hand van de waarde van r één van de subroutines 100, 200, etc. wordt uitgekozen.
Na een druk op toets "ENTER" voert de computer het programma uit voor één worp. Na het gebruik moet u even toets "s" indrukken.

**Speelkaarten en hun problemen**

Tot nu toe hebben we toevalstechnieken gebruikt voor het uitkiezen van een getal of een bepaalde gebeurtenis. Het lijkt er op, alsof we dezelfde technieken ook kunnen gebruiken voor het programmeren van kaartspelletjes. Een kaartspel is opgebouwd uit vier sets van ieder 13 speelkaarten. Kaarten kunnen op diverse manieren door een computer worden getrokken. Het simpelst is de methode, waarbij men twee willekeurige getallen genereert. Een getal tussen 1 en 4, dat de kaartset bepaalt en een tussen 1 en 13, waarmee een bepaalde kaart uit die set wordt gekozen. Het nadeel van dit systeem is dat een en dezelfde kaart diverse malen achter elkaar kan worden getrokken! Net alsof u bij een gewoon kaartspel eerst een kaart zou trekken, onthouden welke kaart dat was en hem nadien weer op zijn plaats kon zetten. Dit is zeer ongebruikelijk. Normaal wordt een kaart slecht eenmaal per spel uitgedeeld. Natuurlijk kan men ook dit programmeren, maar dat is erg ingewikkeld als men met BASIC werkt.
Een tweede bezwaar van deze methode is, dat het resultaat eigenlijk veel te willekeurig is. De volgorde van de kaarten in een spel is namelijk niet zo willekeurig als men wel denkt. Door het schudden van een spel kaarten worden de kaarten wel globaal door elkaar geschud, maar de kans is groot dat twee kaarten in dezelfde volgorde achter elkaar blijven zitten. Geoefende kaartspelers maken van dit gegeven gebruik en proberen de volgorde van de kaarten bij het vorige spel te onthouden en daar hun voordeel uit te halen. Voor zo'n speler is er niets aan om tegen een comuter te spelen.
Gelukkig kunnen we deze twee problemen oplossen, door de computer een spel kaarten op een bepaalde manier te laten simuleren. We kunnen bijvoorbeeld een reeks van 52 verschillende symbolen opbouwen, één symbool voor iedere kaart. Het uitdelen van het spel kaarten kunnen we dan simuleren door ieder symbool om beurt te laten uitschrijven op het scherm. Het toevalseffect ontstaat, door af en toe de computer opdracht te geven het stel symbolen "door elkaar te schudden".
Voor één kleur ontstaat dan het volgende programma:

```
10 LET a$=" AH 2H 3H 4H 5H 6H 7H 8H
   9H 10H JH QH KH"
20 GOSUB 100
30 LET i=1
40 PRINT a$ (i TO i+2)
50 LET i=i+3
60 IF i= 13*3+1 THEN STOP
70 GOTO 40
100 FOR i=1 TO 13
110 LET j=INT (RND*13)
120 LET b$=a$(j*3+1 TO j*3+3)
130 LET a$=a$(1 TO j*3)+a$(j*3+4 TO)
140 LET a$=a$+b$
150 NEXT i
160 RETURN
```



Regel 10 vormt de computernotatie voor de set kaarten van harten-aas tot en met harten-boer. De subroutine van regel 100 introduceert een eenvoudig "schud"-systeem door willekeurig één kaart uit te kiezen en deze 13 maal naar het einde van het spel te verplaatsen. Door de instructies op regels 30 tot en met 70 worden de kaarten onder elkaar op het scherm geschreven. Dit gaat vrij traag en de manier waarop dit programma de kaarten schudt, is alles behalve ideaal.
Door het tussenvoegen van één regel gaat dit schudden al veel beter:

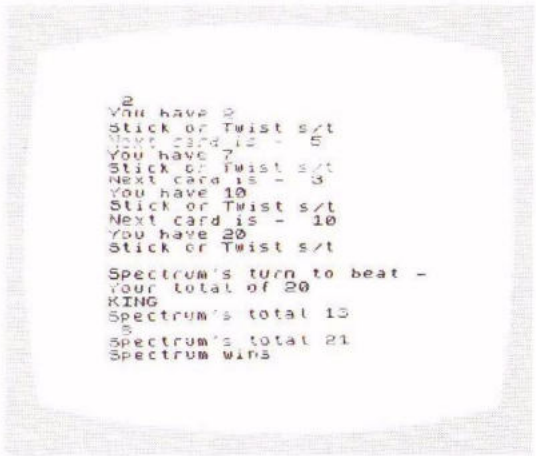
```
25 GOSUB 100
```

Al met al hebben we voor zo iets simpels als het laten schudden van een spel kaarten een vrij uitgebreide subroutine nodig!
Als u zich echter over dat schudden niet al te druk maakt, kunnen we een populair kaartspel als "een-en-twintigen" op de volgende manier programmeren:

```
10 LET t=0
20 LET u=0
30 INK 6:PAPER 1:CLS
40 REM speel een kaart
50 GOSUB 400
60 LET t=t+c
70 IF t>21 THEN GOTO 300
80 PRINT "U hebt ";t
90 PRINT "Pas of kaart p/s"
100 INPUT a$
110 IF a$="p" THEN GOTO 200
120 PRINT "Volgende kaart is - ";
130 GOTO 40
200 PRINT "Spectrum is aan de beurt - "
210 PRINT "uw totaal is ";t
220 LET a$="Spectrum "
230 GOSUB 400
240 LET u=u+c
250 IF u>21 THEN GOTO 300
260 PRINT "Spectrum's totaal ";u
270 PAUSE 100
280 IF u<t THEN GOTO 230
290 PRINT FLASH 1;"Spectrum wint ":GOTO 330
300 IF t>21 THEN PRINT "U hebt";
310 IF u>21 THEN PRINT "Spectrum heeft ";
320 PRINT "verloren"
330 INPUT "volgende spel,druk op ENTER";a$
340 GOTO 10
400 REM neem een kaart
410 LET c=INT(RND*13)+1
420 LET a$=" "+STR$(c)
430 IF c=1 THEN LET a$="AAS"
440 IF c=11 THEN LET a$="BOER"
450 IF c=12 THEN LET a$="VROUW"
460 IF c=13 THEN LET a$="HEER"
470 PRINT a$
480 RETURN
```

Uiteraard is dit een zeer vereenvoudigde versie van één-en-twintigen", voornamelijk opgezet om de toegepaste technieken te verduidelijken.

Dit programma schrijft het spel als volgt op het scherm:




Aan de niet genummerde kaarten worden de volgende waarden toegekent:
AAS = 1
HEER = 13
VROUW = 12
BOER = 11
Alle kaarten worden slechts eenmaal gebruikt, zonder rekening te houden met de kleur, door de "LET"-instructie op regel 410. Regel 420 zorgt er met zijn "STR"-instructie voor, dat zowel de getallen als de woorden (AAS, HEER, etc) op het scherm kunnen verschijnen.

**Ongelijke kansen met een geavanceerde methode**

We kunnen een speciale eigenschap van de Spectrum gebruiken om de methode voor het genereren van willekeurig gehele getallen te perfectioneren, zodat we ze ook kunnen gebruiken als we met gebeurtenissen te maken krijgen, die niet dezelfde kansen hebben. Zoals we weten, kunnen we de volgende formule gebruiken, als we vier verschijnselen gelijke kansen willen geven:

```
10 LET r=INT(RND*4)+1
```

Dit programma kunnen we echter niet gebruiken als we te maken krijgen met verschijnselen die niet helemaal even veel kans mogen krijgen,

zoals bijvoorbeeld bij het weerprogramma, maar een programma als:

```
10 LET r=RND
20 LET w=(r>.4)+(r>.7)+(r>.9)+1
30 PRINT w
40 GOTO 10
```

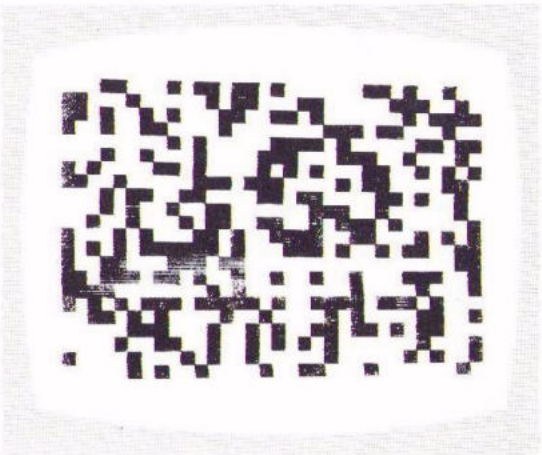
zal cijfers van 1 tot en met 4 op het scherm zetten met dezelfde ongelijke kansen als de verhouding tussen de diverse weersomstandigheden.
In regel 20 worden een aantal ongelijkheden vastgesteld, die de computer één voor één gaat uitwerken. Als de ongelijkheid waar is, bijvoorbeeld r groter dan 0,4 dan interpreteert de computer dat als een ''1''. Is de ongelijkheid niet waar, bijvoorbeeld r kleiner dan 0,4, dan vervangt de computer die ongelijkheid door een ''0''. Laten we bijvoorbeeld r gelijk aan 0,5 stellen en deze waarde invoeren in regel 20. r is groter dan 0,4, dus wordt de ongelijkheid tussen de eerste haakjes (r<P, 4) vervangen door ''1''. Voor alle overige ongelijkheden geldt dat de waarde van r niet voldoet aan het gestelde. De computer vervangt deze ongelijkheden allemaal door ''0''. Als u nu alle resultaten bij elkaar optelt (en de extra '' + 1''-term niet vergeet!) is het resultaat w = 2.
U kunt deze berekening herhalen voor diverse waarden van r (tussen 0 en 1) en steeds zult u constateren dat w gelijk wordt aan het volgnummer van het lijninterval, waarin r valt.
U kunt deze nieuwe kennis gaan misbruiken om de eerder behandelde dobbelsteen- en muntprogramma's te gaan vervalsen, maar dat laten wij geheel voor uw rekening!

**Willekeur en symmetrie**

Zonder enige twijfel heeft u vaak gezien, hoe men met andere computers zeer fascinerende en voortdurend wisselende grafische patronen op een TV-scherm projekteerde. De Spectrum kan dat soort kunstwerken ook aan! Laten we beginnen met een volstrekt willekeurig patroon:

```
10 LET x=INT(RND*32)
20 LET y=INT(RND*22)
30 LET c=INT(RND*8)
40 PRINT AT y,x;INK c;"[ 8]";
50 GOTO 10
```

Let op de manier waarop we grafische symbolen in regel 40 voorstellen, denk aan wat hierover in hoofdstuk 2 is gezegd! Met de instructies op de regels 10 en 20 worden willekeurige plaatsen op het scherm geselecteerd waarop het symbool wordt geschreven. Regel 30 kiest een willekeurige inktkleur.
Op het scherm verschijnt iets dat er ongeveer als volgt uitziet (moment-opname!):



Interessant, zonder meer, maar u krijgt hoofdpijn als u er lang naar kijkt!
Het patroon is namelijk veel te willekeurig. Interessante patronen, die een streling zijn voor het oog, ontstaan alleen maar als er verstandig met willekeur wordt omgesprongen. Zo is een van de basis-principes van de natuur dat alles een bepaalde symmetrie heeft en deze symmetrie moeten we in onze programma's gaan invoeren om orde op zaken te stellen.
Het Spectrum BASIC-dialect geeft ons de mogelijkheid symmetrieën op te bouwen, die ontstaan door als het ware een beeld drie maal te spiegelen. U doet dat het eenvoudigst door een TV-scherm voor te stellen dat door middel van de twee middellijnen in vier even grote delen is verdeeld. Als er in één van die delen een punt wordt geschreven, dan kan men drie symmetrische punten opbouwen door dit ene punt zowel rond de vertikale als rond de horizontale middellijn te spiegelen.
We kunnen de coördinaten van de drie symmetrische punten eenvoudig berekenen, als we er rekening mee houden, dat het volledige scherm is opgebouwd uit 32 bij 22 symboolplaatsen, genummerd respectievelijk 0 tot en met 31 en 0 tot en met 21.

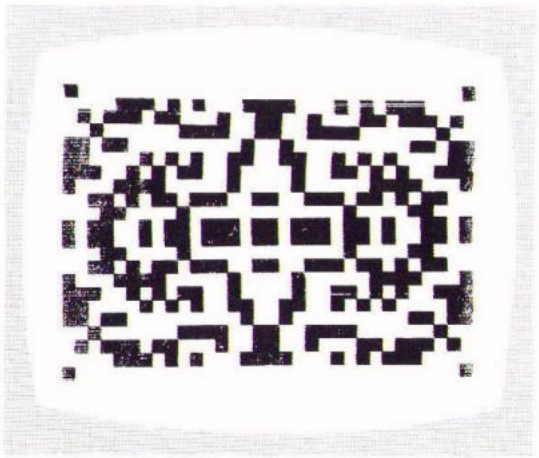
Als de coördinaten van het oorspronkelijk punt "X, Y" zijn, dan zijn de coördinaten van de drie gespiegelde punten: "31-X, Y", "X, 21-Y" en "31-X, 21-Y".
U kunt dat proefondervindelijk aantonen door een vel papier te verdelen in 32 bij 22 vierkantjes, eerst de twee middellijnen en een punt te tekenen en nadien de drie symmetrische punten.
Nu we dit weten, rolt een programma voor een caleidoscoop als het ware uit de pen:

```
10 LET m=31
20 LET n=21
30 LET x=RND*m/2
40 LET y=RND*n/2
50 LET c=INT(RND*8)
60 INK c
70 PRINT AT y,x;"[^8]";
80 PRINT AT y,m-x;"[^8]";
90 PRINT AT n-y,x;"[^8]";
100 PRINT AT n-y,m-x;"[^8]";
110 GOTO 30
```

Met de regels 30 en 40 bepalen we de positie van het eerste punt. Regel 50 definiëert de inktkleur, die door regel 60 wordt vastgelegd voor alle "PRINT"-instructies in de regels 70 tot en met 100.
Dit basis-idee van "vier-quadrant"-symmetrie kan door verdere uitwerking zeer interessante prentjes opleveren. Zo kunnen we bijvoorbeeld het patroon opbouwen door willekeurig vierkantjes te tekenen, beginnende in het midden van het scherm en het beeld nadien naar de randen toe uit te breiden.
Het programma luidt:

```
10 LET m=31
20 LET n=21
30 FOR x=0 TO m/2
40 LET y=RND*n/2
50 LET c=INT(RND*8): INK c
60 PRINT AT y,x;"[^8]";
70 PRINT AT y,m-x;"[^8]";
80 PRINT AT n-y,x;"[^8]";
90 PRINT AT n-y,m-x;"[^8]";
100 NEXT x
110 GOTO 30
```

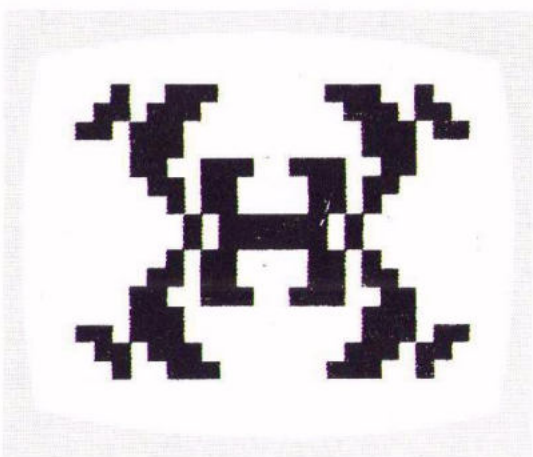
Het is niet mogelijk om wat er op het scherm ontstaat, in dit boek vast te leggen, omdat het voortdurend beweegt en verandert. Wat u hieronder getekend ziet, is echter een goede benadering in zwart-wit van wat er zich op het scherm in kleur afspeelt.
Symmetrische willekeur kunnen we ook nog voor andere zaken gebruiken dan alleen maar voor vlokkenpatronen. Als we bijvoorbeeld starten met een punt in het midden van het



scherm en dit punt in alle mogelijke richtingen uitbouwen door bij de punt-coördinaten steeds, bepaald door het toeval, -1,0 óf 1 op te tellen, dan ontstaat er een willekeurig kronkelende lijn. Als we dan bovendien een "vier-quadrant"-symmetrie in het programma inbouwen, kunnen we een verzameling fascinerende beelden verwachten.
Een programma als:

```
10 LET m=31
20 LET n=21
30 LET x=m/2
40 LET y=n/2
50 GOSUB 100
60 LET x=x+RND*2-1
70 LET y=y+RND*2-1
80 GOTO 50
100 PRINT AT y,x;"[^8]";
110 PRINT AT y,m-x;"[^8]";
120 PRINT AT n-y,x;"[^8]";
130 PRINT AT n-y,m-x;"[^8]";
140 RETURN
```

schildert dit soort prentjes op het scherm:



# 4. GRAFISCHE MOGELIJKHEDEN MET HOGE RESOLUTIE

In hoofdstuk 2 zijn de basisprincipes beschreven, waarmee de Spectrum grafieken op het scherm tekent en zijn de twee puntsoorten voorgesteld, waaruit het scherm bestaat: inkt- en papierpunten. Door de symboolplaatsen van het scherm op te vullen met symbolen en vormen kunnen we indrukwekkende en tamelijk gedetailleerde grafieken samenstellen. Maar als u van plan bent fijne lijndiagrammen of -omtrekken samen te stellen, dan moet u wél in staat zijn schermpunten onafhankelijk van hun symboolplaats te kunnen beïnvloeden.
Dát is het koninkrijk van de hoge resolutie!
Met de "PLOT"-instructie kunnen we gelijk welk punt op het scherm of inkt- of papierpunt laten zijn, zonder dat we daarmee iets veranderen aan gelijk welk ander punt.
Met de "DRAW"-instructie kunnen we rechte of gebogen lijnen tekenen op iedere gewenste plaats op het scherm.
Waarschijnlijk de meest geavanceerde hoge resolutie-instructie is "CIRCLE" waarmee we, hoe bestaat het, een cirkel met een gegeven straal op het scherm kunnen toveren.
Voor een laaggeprijsde machine als de Spectrum hebben we dus de beschikking over een zeer indrukwekkende set instructies voor hoge resolutie.
Maar, helaas, er doemen enige problemen op als we alle kleurmogelijkheden van de machine willen gaan gebruiken bij het programmeren van grafische voorstellingen met hoge resolutie.

**Het bepalen van een punt**
Als we met de genoemde instructies in staat zijn de eigenschappen van een bepaald punt te programmeren, dan moeten we natuurlijk ook bij machte zijn de plaats van dat ene punt ondubbelzinnig vast te leggen.
Dat gaat vrij eenvoudig, door het punt te definiëren met zijn kolom- en regelnummer. Het kolomnummer wordt meestal de x-coördinaat genoemd en bijgevolg gaat het regelnummer als y-coördinaat door het leven.
Beide coördinaten starten bij nul en omdat de Spectrum 256 horizontale en 176 vertikale punten heeft, variëren de x-coördinaten tussen 0 en 255 en de y-coördinaten tussen 0 en 175.
Wel moeten we er steeds aan denken dat y aan de onderkant van het scherm begint en dat dit de omgekeerde richting is van de tot nu toe gebruikte notatie voor de regelnummers door middel van een "PRINT"-bevel.
Voor wie enige wiskundige achtergronden heeft, zal dit alles zeer logisch zijn. Wie die achtergronden mist, moet onthouden dat het punt met de coördinaten "0,0" in de linker onderhoek van het scherm wordt geschreven.

**De instructies voor het werk met hoge resolutie**
Alvorens we ons gaan bezighouden met de vraag hoe we de technieken met hoge resolutie in programma's kunnen gebruiken, gaan we eerst de drie instructies gedetailleerd behandelen.
Het basisbevel:

```
PLOT x,y
```

verandert het punt met de coördinaten "x,y" in een inktpunt. Zowel de x als y kunnen wiskundige uitdrukkingen zijn, de enige beperking is dat beide getallen binnen het scherm moeten vallen. Met andere woorden, x moet tussen 0 en 255 liggen en y tussen 0 en 175. Is dat niet het geval, dan zal de machine een foutmelding op het scherm zetten.
Het volgende programma brengt de "PLOT"-instructie tot leven:

```
10 INPUT x,y
20 PLOT x,y
30 GOTO 10
```

Het hele scherm wordt volgeschreven door:

```
10 FOR x=0 TO 255
20 FOR y=0 TO 175
30 PLOT x,y
40 NEXT y
50 NEXT x
```

Hoe we met een "PLOT"-instructie een punt in een papierpunt kunnen veranderen, komt na de

introductie van de overige instructies aan de orde.
De "DRAW"-instructie wordt in samenspel met het "PLOT"-bevel gebruikt voor het tekenen van lijnen.
Het bevel:

```
DRAW x,y
```

zal een lijn trekken, die start bij de coördinaten van het laatst getekende punt en eindigt bij het nieuwe punt, waarvan de coördinaten x punten horizontaal en y punten vertikaal gelegen zijn van de start. Het gegeven dat het startpunt afhankelijk is van een vorige instructie en het eindpunt van de lijn niet impliciet wordt gegeven, lijkt niet bij te dragen tot de praktische bruikbaarheid van de "DRAW"-instructie.
In de praktijk sluit dit echter zeer goed aan bij de manier waarop met hoge resolutie wordt gewerkt.
Het gecombineerde bevel:

```
PLOT x,y:DRAW xd,yd
```

tekent een lijn tussen de punten "x,y" en x+xd, y+yd".
Enige simpele berekeningetjes maken duidelijk dat:

```
PLOT x1,y1:DRAW x2-x1,y2-y1
```

een lijn tekent tussen de punten met coördinaten "x1, y1" en "x2, y2".
We gaan verder! Toets het volgende programma in:

```
10 PLOT 100,50
20 DRAW 25,25
```

Er wordt een punt geschreven op "100,50" en daarna een lijn naar rechts en naar boven.
Maar het programma:

```
10 PLOT 100,50
20 DRAW -25,-25
```

daarentegen, tekent vanaf hetzelfde startpunt een lijn naar links en naar onder.
Hierdoor hebben we een belangrijk verschil ontdekt tussen de "PLOT"- en "DRAW"-instructie. Het intoetsen van negatieve getallen heeft geen enkele zin bij een "PLOT"-instructie. Door het gebruiken van negatieve getallen kunnen we, met behulp van de "DRAW"-instructie lijnen trekken van rechts naar links of van boven naar onder.

Wat gebeurt er als we twee "DRAW"-bevelen achter elkaar geven? Waar start dan de tweede lijn? De tweede lijn start gewoon bij het eindpunt van de eerste lijn, zoals in feite wel was te verwachten. U kunt zich daarbij een onzichtbare grafische cursor voorstellen, die over het scherm beweegt en het schrijven van de punten volgt.
Na het uitvoeren van een instructie blijft deze grafische cursor staan op de plaats waar de computer het laatste punt heeft geschreven. We kunnen dus stellen dat een "DRAW"-instructie altijd begint met het tekenen van een lijn vanaf de plaats waar die ingebeelde cursor zich bevindt.
Voor het tekenen van omtrekken, kunt u diverse "DRAW"-instructies achter elkaar geven.
Zo kunt u met het volgende programma een vierkant op het scherm laten schrijven:

```
10 PLOT 50,100
20 DRAW 50,0
30 DRAW 0,-50
40 DRAW -50,0
50 DRAW 0,50
```

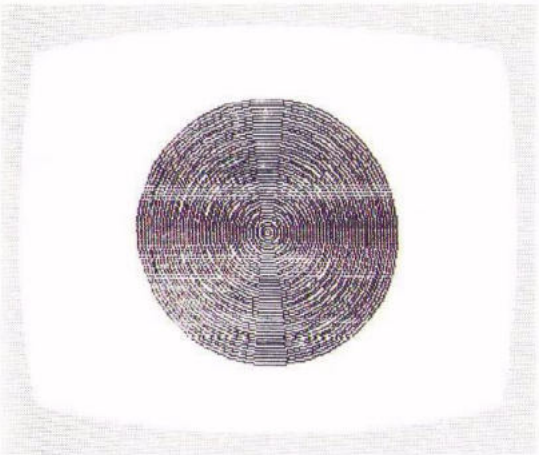
Met het bevel op regel 10 leggen we het beginpunt van de eerste lijn vast op "50,100". De "DRAW"-instructies tekenen één voor één de vier zijden van het vierkant.
Het laatste bevel is:

```
CIRCLE x,y,r
```

en zal een cirkel op het scherm laten tekenen, of althans de beste benadering van een cirkel waartoe de Spectrum in staat is met zijn 256 bij 176 puntenmatrix.
Het middelpunt van de cirkel heeft als coördinaten "x, y" en de straal is r punten lang.
U kunt de "CIRCLE"-instructie uitproberen door het volgende programma in te toetsen:

```
10 FOR r=0 TO 80 STEP 2
20 CIRCLE 128,80,r
30 NEXT r
```

Dit programma schrijft een aantal concentrische cirkels op het scherm, met als gemeenschappelijk middelpunt "120, 80".



Zoals u ziet is dit bevel zeer gemakkelijk in praktijk te brengen. Het enige waar u op moet letten is, dat de cirkel niet gedeeltelijk buiten het scherm valt.

**De "INVERSE"- en "OVER"-instructies en de "attributen"**

Ook bij het programmeren met hoge resolutie kunnen we gebruik maken van de reeds behandelde instructies "INVERSE" en "OVER" en van de attributen-instructies "INK", "PAPER", "FLASH" en "BRIGHT". Het is niet zo moeilijk in te schatten wat voor gevolgen deze instructies hebben, zolang u maar voor ogen houdt dat deze functies invloed hebben op alle punten van een symboolplaats in plaats van op de individuele schermpunten. U kunt deze instructies op dezelfde manier verwerken in bevelen met hoge resolutie als u deed bij het gebruik van "PRINT". Zoals we zullen zien, kunnen zij echter af en toe onverwachte en ongewenste gevolgen hebben.

Laat ons starten met het behandelen van de "INVERSE"- en "OVER"-instructies, deze zijn immers de eenvoudigste.

Als u een punt programmeert, gevolgd door een "INVERSE"-instructie, dan zult u een papierin plaats van een inktpunt generen.

Deze stelling gaat op voor alle grafische instructies. Als u dus een cirkel met inktpunten op het scherm heeft getekent en u wil deze weer weg hebben, dan moet u

```
CIRCLE INVERSE 1,x,y,r
```

intoetsen en de inktpunten van de cirkel worden vervangen door papierpunten.

Met andere woorden, we kunnen de "PLOT INVERSE 1,x,y" -instructie beschouwen als een wisbevel voor een punt.

Als we een met een "DRAW"-bevel getekende lijn willen wissen met een "INVERSE"-instructie, moeten we er wel op letten dat we precies dezelfde lijn-coördinaten inprogrammeren. Als u bijvoorbeeld een lijn had getekend met de instructie:

```
10 PLOT 10,10:DRAW 100,100
```

dan moet u deze lijn wissen met de volgende instructie:

```
10 PLOT 10,10:DRAW INVERSE 1,100,100
```

en niet met:

```
10 PLOT 100,100:DRAW INVERSE 1,-100,-100
```

Het uitwissen start dan immers bij het eindpunt van de originele lijn en of dat altijd goed gaat, is maar af te wachten! Om er zeker van te zijn dat ieder punt wordt gewist, moet u altijd in dezelfde richting wissen als voorheen was geschreven.

Ook het "OVER"-bevel kan uitstekend in programma's met hoge resolutie worden opgenomen. Daar deze bevelen normaal gesproken slechts inktpunten produceren, is de werking van het "OVER"-bevel erg simpel. Een bevel met hoge resolutie, gevolgd door een "OVER"-instructie, zet een bestaande inktpunt om in een papierpunt en een bestaande papierpunt om in een inktpunt.

Als u een "OVER 1"-bevel laat volgen door een "INVERSE 1"-instructie, zulen beide bevelen elkaar opheffen.

Het bevel "INVERSE "1 zorgt dat papierpunten worden geproduceerd door bevelen met hoge resolutie. Denk er echter wel aan dat er, als gevolg van de eigenschappen van de "OVER 1"-instructie, een inktpunt wordt geschreven als

we een papierpunt programmeren op de plaats waar zich reeds een inktpunt bevond! Net zoals er trouwens een papierpunt ontstaat, als we een papierpunt programmeren op de plaats waar reeds een papierpunt aanwezig was.
Dat verklaart waarom er met het bevel:

```
PLOT INVERSE 1,OVER 1,x,y
```

niets op het scherm verandert.
Overigens wil dit niet zeggen, dat dit bevel geen toepassingen heeft. We kunnen dit bevel bijvoorbeeld gebruiken om de onzichtbare grafische cursor naar een bepaald punt van het scherm te sturen, zonder dat er iets op het scherm wordt geschreven.
Omdat de attributen altijd betrekking hebben op alle punten van een symboolplaats, is het gebruik van deze instructies in programma's met hoge resolutie vrij gecompliceerd. Het kan; maar u mag niet vergeten dat alle punten van de symboolplaats, waarin zich dat ene punt bevindt, zullen worden aangetast, zodra er een bevel met hoge resolutie wordt geprogrammeerd. Zo kan het gebeuren dat als u één punt bijvoorbeeld groen maakt, alle andere punten rond dit ene punt ook groen worden omdat ze deel uitmaken van dezelfde symboolplaats. Hetzelfde geldt voor het programmeren van de kleur van papierpunten, voor knipperende punten en voor punten waarvan de intensiteit wordt verhoogd. We kunnen dit toelichten aan de hand van een voorbeeld:

```
10 PLOT INK 3,10,10
20 DRAW INK 3,100,100
30 PLOT INK 5,10,100
40 DRAW INK 5,PAPER 2,100,-100
```

De twee eerste regels tekenen een rode diagonaal over het scherm. De regels 30 en 40 tekenen een tweede diagonaal, die de eerste snijdt. Onmiddellijk vallen twee feiten op.
Om te beginnen zal het duidelijk zijn dat als twee verschillend gekleurde lijnen elkaar snijden, het snijpunt slechts één kleur kan hebben. Uit het voorbeeld wordt duidelijk, dat dit de laatst geschreven kleur is, dus kleur 5, cyaan.
Op de tweede plaats stellen we vast dat de "PAPER"-instructie in regel 40 alle symboolplaatsen beïnvloedt die door de getekende lijn worden doorlopen. En dat ondanks het feit dat het "DRAW"-bevel enkel inktpunten tot gevolg heeft!
Dit voorbeeld toont overduidelijk aan, dat we zeer voorzichtig moeten zijn met het gebruik van attribuut-instructies in programma's met hoge resolutie. Onervaren programmeurs kunnen zich dan ook het best beperken tot twee kleuren bij het werken met hoge resolutie.
Als u tóch alle kleuren van de Spectrum wil toepassen, moet u de prentjes zo ontwerpen dat er nooit meer dan twee kleuren elkaar snijden. Meestal is dat onmogelijk, tenzij u genoegen neemt met tamelijk simpele plaatjes.
Bovendien moet u bedenken dat twee kleuren voldoende zijn om die beelden te verlevendigen, die zich het beste met technieken met hoge resolutie laten opbouwen, zoals bijvoorbeeld grafieken.

**Cirkels en ellipsen**

Met het "CIRCLE"-bevel kunnen we op een zeer eenvoudige manier een cirkel op het TV-scherm projecteren. Daarnaast kunnen we natuurlijk ook een cirkel schrijven, door gebruik te maken van de wiskundige principes, die aan de basis liggen van de cirkelvorm. Dat heeft als voordeel dat we dan ook ellipsen kunnen tekenen, want een ellips is niets anders dan een vervormde cirkel en gehoorzaamt dezelfde wiskundige wetten. Vergeet niet, dat een ellips een van de basisvormen is, die we nodig hebben voor het opbouwen van drie-dimensionele beelden!
De coördinaten van alle punten op de omtrek van een cirkel met als middelpunt "x1, y1" en straal r, voldoen aan de volgende twee uitdrukkingen:

```
x=r*SIN(t)+x1
```

en:

```
y=r*COS(t)+y1
```

voor een bepaalde waarde van t.
U hoeft de wiskundige achtergronden van deze vergelijkingen niet te begrijpen om er praktisch mee te kunnen werken! Het belangrijkste om te onthouden is dat als u aan t een bepaalde waarde toekent en deze in beide formules verwerkt, er een punt ontstaat dat ligt op de omtrek van de cirkel met straal r en middelpunt op "x1, y1".

Dat wordt door het volgende programma aangetoond:

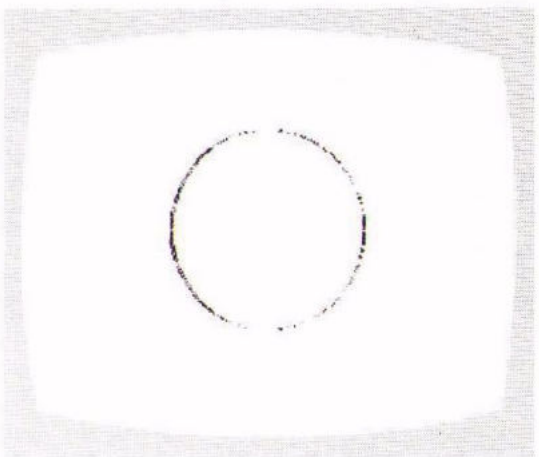
```
10 LET r=60
20 LET x1=100
30 LET y1=60
40 LET t=RND* 283
50 PLOT r*SIN(t)+x1,r*COS(t)+y1
60 GOTO 40
```

Er verschijnen in een willekeurige volgorde punten op het scherm, die de omtrek van een cirkel vormen. Door de "RND"-instructie in regel 40 geven we aan t een willekeurige waarde en daardoor ontstaat er een willekeurig punt op de cirkelomtrek.
Als we er zin in hebben, kunnen we aan t de beginwaarde 0 toekennen en nadien, door de waarde van t langzaam te laten stijgen, alle punten van de cirkel achter elkaar op het scherm zetten. Als we het programma ook nog de waarde van t op het scherm laten schrijven, zullen we vaststellen dat de cirkel wordt gesloten als t gelijk is aan 6,283, wat niet toevallig gelijk is aan $2 \times \pi$ (pi).
Iedereen die iets van wiskunde afweet, zal begrijpen hoe dat komt, maar het is niet zo belangrijk dat we er van wakker hoeven te liggen. In plaats van de ruwe benadering kunnen we gebruik maken van de PI-toets van de computer:

```
10 LET r=60
20 FOR t=0 TO 2*PI STEP .01
30 PLOT r*SIN t+100,r*COS t+80
40 NEXT t
```

Er verschijnt nu een cirkel op het scherm, maar wel veel trager dan wanneer we gebruik zouden maken van de "CIRCLE"-instructie. Door het invoeren van diverse "STEP"-waarden kunnen we het aantal punten bepalen, waarmee de omtrek wordt getekend.
Door gebruik te maken van de "SIN" en "COS"-functies kunnen we cirkels tekenen met de hoogst mogelijke graad van nauwkeurigheid. Een nauwkeurigheid die alleen wordt begrensd door de allesbehalve oneindige resolutie van de 256 bij 176 punten van het scherm van de Spectrum.
Hoe de computer de punten berekent als we gebruik maken van de "CIRCLE"-instructie wordt niet uitgelegd in de bij de machine horende handleiding. Vandaar dat we er niet zeker van kunnen zijn of deze manier van cirkels tekenen wel zo nauwkeurig is. Door een simpel programmaatje kunnen we beide methoden met elkaar vergelijken. Het onderstaande programma tekent eerst een cirkel volgens de "SIN/COS"-formule en daarna dezelfde cirkel met de "CIRCLE"-instructie. Door het invoeren van een "OVER 1"-bevel zullen alle punten van beide cirkels die op dezelfde locaties ontstaan, tegen elkaar wegvallen, met andere woorden: op die plaatsen worden papierpunten geschreven. Wat dus wél op het scherm verschijnt zijn die punten van de cirkels die niet samenvallen, dus die punten waarbij de computer volgens de ene methode andere coördinaten berekent dan volgens de andere.

```
10 LET r=60
20 FOR t=0 TO 2*PI STEP .01
30 PLOT r*SIN t+100,r*COS t +80
40 NEXT t
50 OVER 1
60 CIRCLE 100,80,60
```

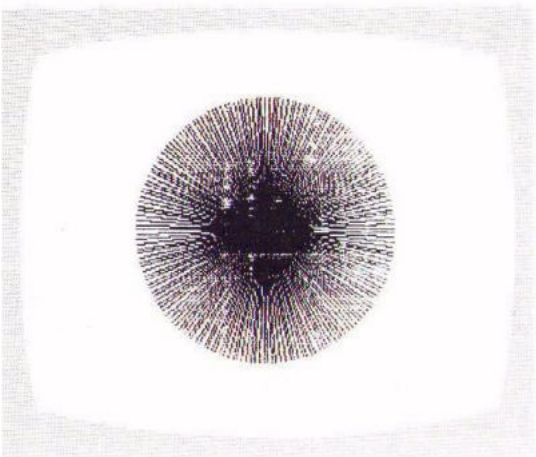


Uit deze afbeelding volgt, dat er nogal wat punten zijn, waarbij dit het geval is. Toch zijn de afwijkingen erg klein en aanvaardbaar als men rekening houdt met de snelheid waarmee een "CIRCLE"-bevel wordt uitgevoerd.
Met de kennis van de coördinaten van punten op een cirkelomtrek kunnen we ook andere dingen doen dan enkel een cirkel tekenen!
Door bijvoorbeeld lijnen te tekenen vanuit het middelpunt naar de omtrek, kunnen interessante patronen ontstaan:

```
10 LET r=80
20 LET c=INT(RND*8)
30 INK c
```

```
40 PAPER 9
50 CLS
60 FOR t=0 TO 2*PI STEP (RND+RND)/10
70 LET x=r*SIN t
80 LET y=r*COS t
90 PLOT 128,80
100 DRAW x,y
110 NEXT t
120 PAUSE 100
130 GOTO 20
```

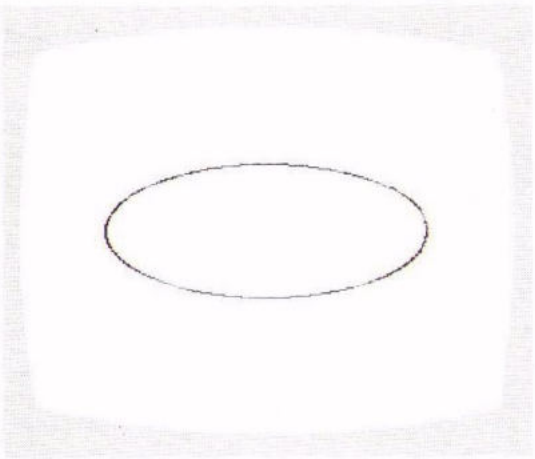
Met regel 90 sturen we de grafische cursor naar het middelpunt van de cirkel. Regel 100 is verantwoordelijk voor het tekenen van de lijnen naar de omtrek van de cirkel.
Let op het gebruik van de "kleur 9" (zie hoofdstuk 2) voor het automatisch bepalen van de achtergrondkleur voor maximaal contrast.



Nu u weet hoe u cirkels kunt tekenen, is het niet moeilijk deze methode uit te breiden naar het tekenen van ellipsen. Zoals u wellicht weet, is een ellips een "afgeplatte" cirkel. Een ellips is wat u ziet, als u onder een afwijkende hoek naar een cirkel kijkt.
Vandaar dat het kunnen tekenen van ellipsen zo belangrijk is voor het samenstellen van driedimensionale beelden.
Een ellips ontstaat, als u in de twee formules van de sinus-functie twee verschillende waarden voor de straal invoert: een horizontale en een vertikale straal.
Probeer maar eens uit!

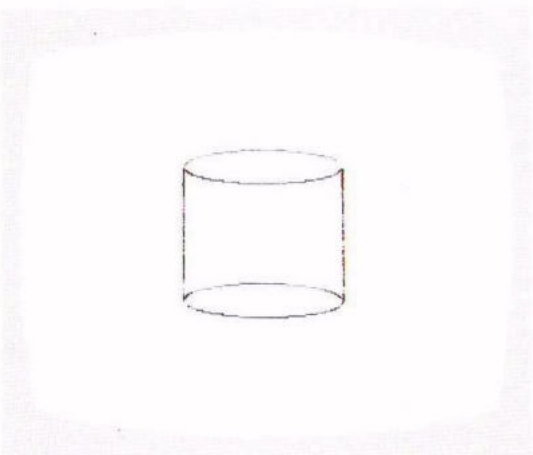
```
10 LET r1=100
20 LET r2=40
30 FOR t=0 TO 2*PI STEP .01
40 PLOT r1*SIN t +128,r2*COS t +80
50 NEXT t
```

Zoals u ziet, tekent dit programma een ellips op het scherm, met een horizontale straal van 100 punten lang en en vertikale van 40 punten lang. Door andere getallen voor r1 en r2 in te toetsen, ontstaan ellipsen met uiteenlopende verhoudingen.



Als simpel voorbeeld hoe we met ellipsen driedimensionale vormen kunnen tekenen, geven we een programma voor het tekenen van een soepblik!

```
10 FOR t=0 TO 2*PI STEP .02
20 PLOT 50*SIN t+128,10*COS t+100
25 PLOT 50*SIN t+128,10*COS t+20
30 NEXT t
40 PLOT 128+50,20
50 DRAW 0,80
60 PLOT 128-50,20
70 DRAW 0,80
```



Na het voorgaande moet u in staat zijn de twee ellipsen in dit programma terug te vinden.

**Een spel met pijlen**

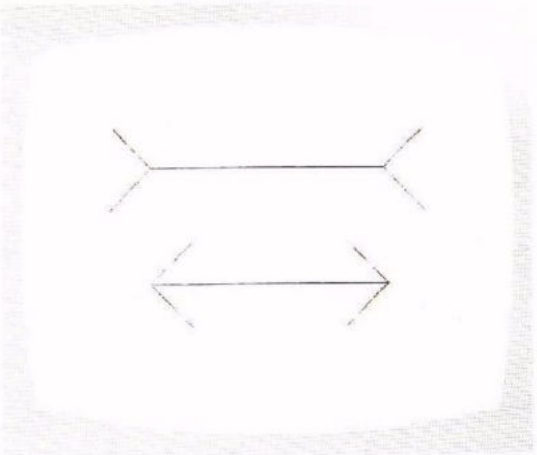
Het pijlenspel is een eenvoudig voorbeeld van het werken met hoge resolutie. Er worden twee pijlen met ieder een verschillende, door het toeval bepaalde lengte, op het scherm getekend. De lengte van de tweede pijl kan worden gevariëerd door het indrukken van de toets " ← ", waardoor de pijl korter wordt en door het bedienen van de toets " → ", waardoor de pijl langer wordt. Het doel van het spel is beide pijlen precies even lang te maken. Dit lijkt erg simpel, tot u het in de praktijk probeert en een wel bekend optisch bedrog toepast.

Het programma is vrij lang:

```
10 LET L=INT((RND*20)+140)
20 PLOT 50,120
30 DRAW L,0
40 PLOT 50,120
50 DRAW -25,25
60 PLOT 50,120
70 DRAW -25,-25
80 PLOT 50+L,120
90 DRAW 25,25
100 PLOT 50+L,120
110 DRAW 25,-25
120 LET s=INT((RND*50)+100)
130 GOSUB 500
140 LET a$=INKEY$
150 IF a$=""THEN GOTO 140
160 IF a$="s"THEN GOTO 300
170 IF a$="5"THEN GOSUB 500:LET s=s-1:
    GOTO 130
180 IF a$="8"THEN GOSUB 500:LET s=s+1:
    GOTO 130
190 GOTO 140

300 PRINT AT 18,0;"Bovenkant pijltje=";L
310 PRINT AT 19,0;"Onderkant pijltje=";s
320 PRINT AT 20,0;"Een verschil van"
330 PLOT 50,4
340 DRAW ABS(L-s),0
350 IF L=s THEN  PLOT OVER 1;50,4
360 STOP

500 OVER 1
510 PLOT 50,50
520 DRAW s,0
530 PLOT 50,50
540 DRAW 25,25
550 PLOT 50,50
560 DRAW 25,-25
570 PLOT 50+s,50
580 DRAW -25,-25
590 PLOT 50+s,50
600 DRAW -25,25
610 OVER 0
620 RETURN
```

Door middel van de regels 10 tot en met 110 wordt de eerste pijl op het scherm getekend. De instructie op regel 120 bepaalt bij toeval de lengte van de tweede pijl, die nadien door de subroutine 500 op het scherm wordt getekend.

Let op het gebruik van het "OVER 1"-bevel! Het eerste beroep op deze subroutine heeft tot gevolg dat de tweede pijl wordt getekend, een tweede gebruik van de routine doet de pijl weer



verdwijnen. Door de regels 140 tot 180 weet de computer of een van de toetsen wordt ingedrukt. Een druk op de "rechter pijl"-toets laat de pijl langer worden en tekent hem nadien opnieuw op het scherm, een druk op de "linker pijl"-toets reduceert de lengte van de pijl en print hem nadien weer uit.

Na het drukken op de "s"-toets verschijnen de echte pijllengten en het eventuele lengteverschil op het scherm, dat alles onder bevel van de regels 300 tot en met 350.

Ondanks de eenvoud van het programma, zien we toch zeer duidelijk hoe de lengte van de tweede pijl reageert op het bedienen van een van beide toetsen!

**Een te hoge resolutie**

Beginnende programmeurs maken vaak te overdadig gebruik van technieken met hoge resolutie. We hebben reeds diverse malen gesteld dat heel wat tekeningen gemakkelijker en zelfs beter op het scherm zijn te zetten door gebruik te maken van technieken met lage resolutie.

Neem als voorbeeld de programma's uit hoofdstuk 3, waarmee we willekeurige symmetrische patronen op het scherm hebben geschreven.

Nu u de technieken voor het programmeren met hoge resolutie onder de knie hebt, wilt u waarschijnlijk deze programma's herschrijven door instructies met hoge resolutie, die immers in staat zijn ingewikkelder patronen te verwezenlijken.

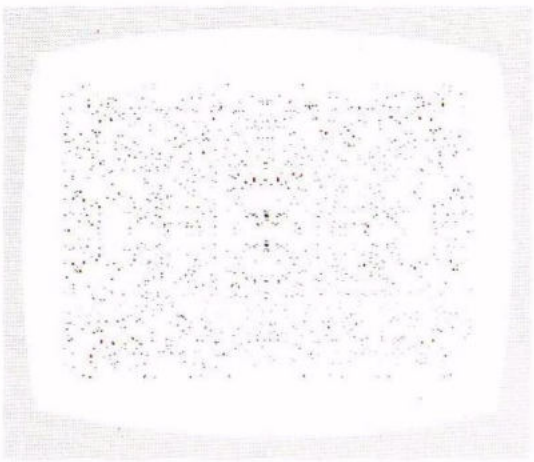
Doe het niet!

In de eerste plaats kunt u dan maar twee kleuren

gebruiken, terwijl de programma's met lage resolutie alle acht kleuren een kans geven.
Zelfs als u genoegen neemt met die twee kleuren, zult u toch zeer teleurgesteld zijn over het resultaat.
Probeer maar eens het volgende programma:

```
10 OVER 1
20 LET x=INT (RND*128)
30 LET y=INT (RND*88)
40 PLOT x,y
50 PLOT 255-x,y
60 PLOT x,175-y
70 PLOT 255-x,175-y
80 GOTO 20
```

Met dit programma wordt een zeer ingewikkeld patroon getekend, zo ingewikkeld, dat het effect verdwenen is. De puntjes zijn veel te klein - de resolutie veel te hoog!
Kijk maar!



**Een geavanceerde vorm van het "DRAW"-bevel**

De laatste paragraaf van dit hoofdstuk behandelt een tamelijk gecompliceerde vorm van de "DRAW"-instructie en kan zonder nare gevolgen voor het begrijpen van de volgende hoofdstukken worden overgeslagen.
De volledige uitdrukking van een "DRAW"-bevel luidt:

```
DRAW x,y,t
```

waarin x en y nog steeds hetzelfde betekenen, maar t een hoek voorstelt gemeten in radialen.

Laten we de oude vorm van de "DRAW"-instructie eens vergelijken met de nieuwe notatie.
De oude vorm:

```
10 PLOT x1,y1:DRAW x2-x1,y2-y1
```

tekent een rechte lijn op het scherm tussen de punten "x1, y1" en "x2, y2".
De uitgebreide vorm:

```
10 PLOT x1,y1:DRAW x2-x1,y2-y1,t
```

tekent een lijn tussen dezelfde punten, maar nu geen rechte, maar een deel van een cirkelomtrek. Met andere woorden: er ontstaat een boog.
De lengte van de boog wordt bepaald door de waarde van t. Voor de niet-wiskundig getrainde lezers, die niet vertrouwd zijn met het uitdrukken van hoeken in radialen, kan het volgende tabelletje enig houvast geven.

| waarde t | grootte van de boog |
|---|---|
| 2*PI | volledige cirkel |
| PI | halve cirkel |
| PI/2 | kwart cirkel |

Het probleem met deze instructie is dat we erg moeilijk bij het programmeren kunnen inschatten of de boog die zal worden getekend wel binnen het scherm valt. Toch is "DRAW" dé oplossing, als we een deel van een cirkel moeten tekenen.
Zo zal er een halve cirkel op het scherm verschijnen, als we het onderstaande programma laten uitvoeren:

```
10 PLOT 100,80
20 DRAW 50,50,PI/2
```

Er zijn in principe twee manieren om een boog tussen twee punten te tekenen: linksom en rechtsom. Ook dat kan de Spectrum: verander PI/2 door -PI/2 en kijk wat er gebeurt.
De uitgebreide "DRAW"-instructie wordt niet vaak toegepast en wel om de eenvoudige reden dat het niet vaak voorkomt dat er een boog moet worden getekend.

# 5. GELUID

We kunnen de mogelijkheid van de Spectrum om via zijn kleine, ingebouwde luidspreker geluiden te produceren, gebruiken om spelletjes spannender te maken.
Daarnaast kunt u het geluid van de machine benutten om wat meer over het verschijnsel "muziek" te weten te komen, als u daar wel interesse in hebt, maar noch het geduld, de tijd of de bekwaamheid hebt kunnen opbrengen om een muziekinstrument te leren bespelen.
Toch zijn er een groot aantal problemen en beperkingen aangaande het geluid van de Spectrum.
De machine is monofoon, kan dus slechts één noot tezelfdertijd produceren. Het is niet mogelijk het volume noch de klankkleur van het geluid te beïnvloeden. Daarnaast zijn speciale geluidseffecten tamelijk moeilijk in BASIC te programmeren. Tot slot kan de Spectrum slechts één ding tegelijk doen, dus ofwel geluiden maken, ofwel programma's afhandelen.
Het gevolg van deze laatste beperking is, dat het zeer moeilijk is in BASIC een spel met muzikale begeleiding te programmeren; de machine is dan veel te druk met het verwerken van het spelletjesprogramma om ook nog eens muziek te maken.
Sommige beperkingen kunnen weliswaar door een uitgekiend programma worden omzeild, maar als u na het lezen van dit hoofdstuk meer wilt doen met computermuziek, dan zult u toch moeten gaan nadenken over het volgen van een cursus programmeren in machinecode.
In dit hoofdstuk zult u kennismaken met de geluidsmogelijkheden van de Spectrum met niets meer dan BASIC-programma's.

**De "BEEP"- en "PAUSE"-instructies**
De Spectrum heeft slechts één geluidsinstructie, namelijk "BEEP".
De instructie:

```
BEEP tijdsduur,toonhoogte
```

produceert gedurende een bepaalde tijd een bepaalde toon. Het tijdsduurbevel kan elke waarde worden gegeven en wordt door de computer gemeten in seconden. Ook aan het toonhoogtebevel kan elke waarde worden toegekend, maar de machine vertaalt deze waarde via een vrij ingewikkeld systeem naar halve tonen beneden en boven de middelste C. Dit zullen we later gedetailleerd uitleggen.
De voornaamste eigenschap van een "BEEP"-bevel is dat de computer een aantal seconden bezig is en gedurende die tijd niets anders kan doen.
Als u bijvoorbeeld in een programma stelt, dat de Spectrum tien seconden lang een bepaalde toon moet produceren, kunt u niet verwachten dat er in die periode ook nog iets op het scherm gebeurt! De "BEEP"-instructie lijkt dus in feite op een "PAUSE"-bevel met geluid!
Sommige computers hebben geen problemen met het tezelfdertijd produceren van een toontje en het uitvoeren van bevelen van een programma. Dat is uiteraard een meer bevredigende methode en vandaar dat we op het einde van dit hoofdstuk zullen bespreken hoe we dat ook met de Spectrum kunnen doen.
Met de "BEEP"-instructie kunnen we de machine gedurende een programmeerbare tijd een programmeerbare noot laten spelen, maar als we de computer een melodietje willen laten spelen is het ook nodig dat we rustpauzes kunnen programmeren.
Daarvoor staat de "PAUSE"-instructie ter beschikking, die met:

```
PAUSE tijdsduur
```

de machine dwingt een bepaalde tijd absoluut niets uit te voeren. Toch moeten we twee dingen heel goed in de gaten houden als we de "PAUSE"-instructie samen met een "BEEP"-bevel gebruiken. In de eerste plaats meet de computer de duur van de "PAUSE" niet in hele, maar in vijftigste delen van een seconde!
Gelukkig kunnen we dit probleem gemakkelijk oplossen: het bevel:

```
PAUSE tijdsduur*50
```

zet een in seconden ingevoerde tijd automatisch om in vijftigste delen van een seconde.
Een tweede eigenaardigheid van het "PAUSE"-bevel is dat het onmiddellijk wordt ingetrokken door het indrukken van gelijk welke toets van het toetsenbord. Daar is niets tegen te doen, het is dus opletten geblazen, en tijdens concerten van de Spectrum met de vingers van het toetsenbord afblijven!
Hoe produceert de machine toontjes? Zeer simpel. Zoals u waarschijnlijk wel weet bestaat geluid uit luchttrillingen.
De Spectrum produceert deze luchttrillingen door de conus van het ingebouwde luidsprekertje heen en weer te bewegen. De toonhoogte of frequentie van het geproduceerde geluid hangt af van het aantal trillingen per seconde, dus van het aantal malen dat de conus van de luidspreker heen en weer wordt bewogen. Dat aantal bewegingen is de enige grootheid die we kunnen programmeren. Er zijn maar twee standen van de conus: volledig naar binnen getrokken of volledig naar buiten geduwd. Eén heen-en-weer gaande beweging wekt een luide klik op, te horen als we het volgende programma intoetsen:

```
BEEP .001,0
```

Als we deze beweging steeds sneller achter elkaar herhalen, zullen we op een bepaald moment een continue toon horen, in plaats van afzonderlijke klikjes. Hoe sneller de beweging, hoe hoger de frequentie of de toonhoogte van het geluid.
Dit kunnen we uitproberen aan de hand van de volgende regels:

```
10 FOR i=-60 TO 69
20 BEEP .5,i
30 NEXT i
```

Met dit programma krijgen we bovendien een inzicht in het volledige geluidsbereik van de machine, al zijn niet alle tonen praktisch bruikbaar. De zeer hoge noten klinken tamelijk vals en de zeer lage tonen bepaald onaangenaam.

**Het spelen van blad-muziek**

Niets is eenvoudiger dan het spelen van een melodietje! U hoeft alleen maar de juiste toon op het juiste moment met de juiste lengte op te wekken. Het enige probleem daarbij is dat muziek al veel langer bestaat dan computers en muzikanten hun eigen programmeertaal hebben uitgevonden: muziekschrift! Gelukkig is muziekschrift niet moeilijk te leren en iedereen kan vrij snel muziek leren lezen.
Muziekschrift bevat twee soorten informatie: de toonhoogte en de duur van een bepaald geluid. De toonhoogte van het geluid wordt gesymboliseerd door de plaats op een aantal horizontale lijnen (de notenbalk) waar we een tekentje zetten. Iedere plaats op die notenbalk wordt aangeduid met een bepaalde letter, zie figuur 1.

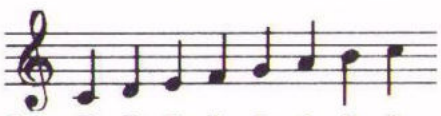


*Fig. 1*

We starten bij de C en gaan dan in alfabetische volgorde tot aan de G, dan springen we naar A en gaan weer naar C.
Dat de C twee keer voorkomt is niet toevallig en heeft alles te maken met het feit dat noten met dezelfde lettercode hetzelfde klinken. Het muzikale bereik tussen twee noten met dezelfde letter, noemen we een "octaaf".
Nu moeten we nog leren hoe we deze noten omzetten in toonwaarden voor onze computer. Wat muzikanten de "middelste C" noemen, correspondeert met een toonwaarde 0. Het verhogen van de toon met een eenheid heeft de eerstvolgende halve noot tot gevolg en hetzelfde geldt voor het verlagen van de toon met één eenheid, de computer wekt dan de vorige halve noot op.
Als dat allemaal waar is, dan lijkt het er op dat we van C naar D gaan door de toon met twee eenheden te verhogen en inderdaad, dat klopt! Het verhogen van de toon met 1 wekt de eerstvolgende halve noot op, het verhogen van de toonwaarde met 2 verhoogt de toonhoogte tot de eerstvolgende hele noot.
Helaas bestaan er uitzonderingen op deze regel, al heeft dat niets te maken met de Spectrum, maar alles met de muziektheorie. Het verhogen van de toon met nog eens twee eenheden heeft tot gevolg dat de computer de E-noot laat horen.
Als we dan echter nog eens 2 eenheden bij de toonwaarde optellen, komen we niet bij de F uit!

Dat komt, omdat er in de westerse muziek niet steeds een hele toon ligt tussen twee noten. Dat blijkt ook uit de verdeling van de witte en de zwarte toetsen op een piano. Als we met de letter T een hele toon en met de letter H een halve toon symboliseren, kunnen we de verdeling van de hele en halve tonen over een octaaf als volgt voorstellen:

$$C^{T}D^{T}E^{H}F^{T}G^{T}A^{T}B^{H}C$$

Tussen de noten E en F en B en C zit maar een halve toon dus moeten we daar de toonwaarde van de Spectrum met slechts één eenheid verhogen!
Ook als u dat vrij ingewikkeld vindt moet u toch door deze zure appel heen bijten. Het is echt belangrijk en dit kunnen we toelichten aan de hand van de volgende twee programma's:

```
10 FOR i=0 TO 14 STEP 2
20 BEEP .5,i
30 NEXT i
```

en:

```
10 DATA 0,2,4,5,7,9,11,12
20 FOR i=1 TO 8
30 READ p
40 BEEP .5,p
50 NEXT i
```

Het eerste programma produceert een toonladder, waarbij er vanuit is gegaan dat er tussen iedere noot een hele toon ligt. Het tweede programma speelt de correcte C-toonladder. Een heel verschil, vindt u ook niet? Zonder enige twijfel vindt u de tweede toonladder veel melodieuzer klinken. En al zou dat niet het geval zijn, de meeste mensen vinden van wél en alle melodietjes zijn volgens deze schaal geschreven.
In het tweede programma introduceren we de "DATA"-instructie, waarmee we de toonwaarden van de door de Spectrum gespeelde noten kunnen vastleggen, een principe dat we veel zullen gebruiken.
Aan de hand van het voorgaande kunnen we een overzichtje tekenen, waarin we het verband tussen de volledige notenbalk voor één octaaf en de verschillende toonwaarden vastleggen.

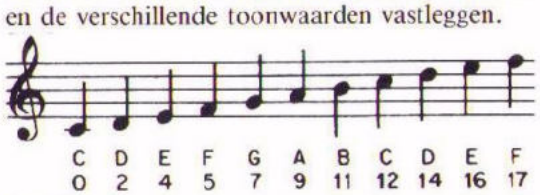


*Fig. 2*

Als u een stuk muziekschrift wil omzetten in toonwaarden hoeft u alleen maar rekening te houden met de in figuur 2 getekende onderlinge relaties. Wat u nog wel moet weten is dat muzikanten soms halve tonen moeten spelen.
Er bestaan twee extra tekens, waarmee we kunnen aanduiden of een bepaalde noot een halve toon hoger of lager moet worden gespeeld: "kruis" (#) geeft een stijging met een halve toon aan, en "mol" (b) een daling van een halve toon.
Als u dus ooit een noot met een kruis of een mol tegenkomt, moet u de toonwaarde van die noot respectievelijk een eenheid verhogen of verlagen.
We kunnen de kruis- en molsymbolen op twee manieren in het muziekschrift verwerken. Ofwel door ze in de buurt van afzonderlijke noten te schrijven, ofwel door ze op te nemen in de "sleutel" aan het begin van een notenbalk. Als de kruis- of molsymbolen worden gebruikt aan het begin van het muziekstuk, dan gelden ze voor alle noten op die lijn (en) van de notenbalk voor het volledige muziekstuk.
Sterker nog, niet alleen vallen alle noten op dezelfde lijnen van de notenbalk onder dat kruis of die mol, ook alle noten met dezelfde lettercode moeten een halve toon worden verhoogd of verlaagd. Met andere woorden: in alle gebruikte octaven moeten we op die bepaalde noot een verschuiving van een halve toon gaan toepassen.
Dit alles geldt dus alleen voor een kruis- of molsymbool aan het begin van het muziekstuk. Staat het symbool bij een bepaalde noot, dan geldt de verschuiving alleen voor die noot.

De tweede soort informatie die in muziekschrift is verwerkt, de duur van iedere noot, is veel eenvoudiger te begrijpen.
Iedere noot wordt verondersteld een veelvoud van een bepaalde basistijd te worden aangehouden. Als we veronderstellen dat een "normale" noot, een zogenaamde kwartnoot, één tijdseenheid duurt, dan betekent een schuin streepje aan een noot, dat de tijdsduur wordt gehalveerd. Daarnaast zijn er noten die langer duren dan de kwartnoot: de halve en de hele noot, de respectievelijk twee en vier maal zo lang worden aangehouden als de kwartnoot.
Een en ander is overzichtelijk voorgeschoteld in

figuur 3, waar u tevens alle symbolen terugvindt voor de diverse soorten noten.
In echte muziekstukken zult u vaak twee of meerdere noten vinden, die door middel van het doorverbinden van hun schuine streepjes zijn gegroepeerd. Deze en de overige frase- en accent-tekens zijn voor ons niet van belang, omdat we het volume van de tonen toch niet kunnen veranderen, als we muzek op de Spectrum programmeren. Wél van belang is dat een punt achter een noot de normale duur van deze noot met de helft verlengd. Zo moet een kwartnoot, gevolgd door een punt, anderhalve tijdseenheid worden aangehouden.
Voor een rust of pauze tussen twee noten gelden dezelfde afspraken. De diverse symbolen voor 1/8, 1/4, 1/2 en een hele pauze vindt u ook in figuur 3.

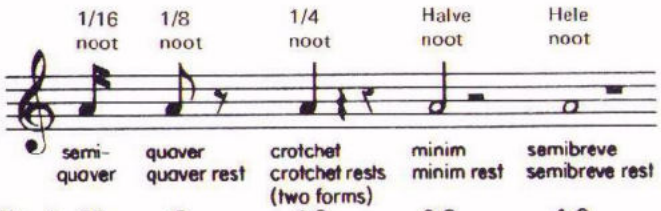


*Fig. 3*

Misschien is het wel handig, onze muzikale kennis in een tabelletje samen te vatten:

| Noot | Toonwaarde | Tijdsduur | |
|---|---|---|---|
| C | 0 | 1/16 noot | 0,25 |
| D | 2 | 1/8 noot of pauze | 0,5 |
| E | 4 | 1/4 noot of pauze | 1,0 |
| F | 5 | 1/2 noot of pauze | 2,0 |
| G | 7 | hele noot of pauze | 4,0 |
| A | 9 | | |
| B | 11 | | |
| C | 12 | | |
| D | 14 | | |
| E | 16 | | |
| F | 17 | | |

**De Spectrum speelt een melodietje**
Na al deze theorie over muzicknotatie is een voorbeeld wel op zijn plaats. De eerste notenbalk van het populaire Amerikaanse deuntje "Dixie" is getekend in figuur 4.

Eerst schrijven we de lettercode van iedere noot onder ieder symbool. Vervolgens zetten we iedere lettercode om in de juiste toonwaarde volgens de tabel, eventueel rekening houdende met kruis- of molsymbolen aan het begin van de notenbalk.
Tot slot de tijdsduur, die u alweer uit het tabelletje kunt overnemen.
Nu moeten we het programma gaan schrijven.

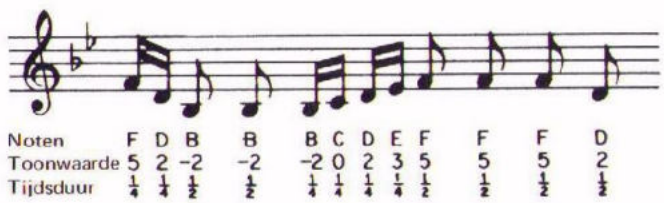


*Fig. 4*

Daarvoor bestaan verschillende systemen, maar een van de meest veelzijdige is het gebruiken van "DATA"-instructies. Ieder koppel gegevens (toonwaarde en tijdsduur) wordt opgenomen in een "DATA"-bevel en het einde van het deuntje wordt aangegeven door een niet in de codelijst opgenomen waarde zoals bijvoorbeeld "99,99". Deze waarde kan dus niet worden vertaald in een toonwaarde of een tijdsduur.

Het volledige programma voor het "Dixie-melodietje wordt:

```
10 DATA 5,.25,2,.25,-2,.5,-2,.5,-2,.25
20 DATA 0,.25,2,.25,3,.25,5,.5,5,.5,5,.5
30 DATA 2,.5,7,.5,7,.5,7,.5,5,.5,7,.5
40 DATA 7,.5,7,.25,9,.5,10,.25,12,.25
50 DATA 14,1.5,10,.25,5,.25,10,1.5,5,.25
60 DATA 14,.25,5,1.5,12,.25,14,.25,10,1.5
70 DATA 5,.25,5,.25,10,.5,14,.5,12,.5
80 DATA 10,.5,7,.5,10,1,10,.5,12,1.5,7,.5
90 DATA 12,1.5,5,.5,10,.5,14,.5,12,.5,10,.5
100 DATA 7,.5,9,.5,10,.75,7,.25,7,.25
```

```
110 DATA 5,.5,10,.5,2,.5,2,.5,0,1,2,.5
120 DATA -2,.75,2,.5,0,.75,7,.5,5,.5,2,.5
130 DATA 10,.75,14,.25,12,.5,10,1,99,99
140 LET d=.5
150 READ p,t
160 IF p=99 THEN STOP
170 BEEP d*t,p
180 GOTO 150
```

Deze techniek kan met gebruiken voor het programmeren van een eenvoudig melodietje. Als er in het programma een pauze moet worden opgenomen, kan men gebruik maken van een voor de computer onbekende waarde voor de toonhoogte, gevolgd door de juiste waarde voor de pauze.

**Het spelen van muziek**

Een heel ander aspect van het programmeren met muziek is het schrijven van een programma, waardoor de Spectrum wordt omgetoverd tot een muziekinstrument. U kunt de computer gebruiken voor het naspelen van bestaande muziek of voor het componeren van eigen melodietjes. Het basis-idee is zeer eenvoudig. Door middel van de "INKEY$"-functie kunt u de computer laten lezen welke toetsen zijn ingedrukt, waarna hij deze vertaalt in tonen welke vooraf aan die toetsen zijn toegekend.
En hierin schuilt nu net het probleem: een toetsenbord van het schrijfmachine-type is niet zo maar te programmeren met alle mogelijke toonhoogtes, noten, halfnoten en pauzes, zodat er een gemakkelijk te bespelen instrument ontstaat.
Toch kunt u, als voorbeeld voor dit soort programma's, het onderstaande intoetsen:

```
10 DATA 0,2,4,5,7,9,11,12
20 DIM n(8)
30 FOR i=1 TO 8
40 READ n(i)
50 NEXT i
60 LET a$=INKEY$
70 IF a$="" THEN GOTO 60
80 BEEP .2,n(VAL(a$(1)))
90 GOTO 60
```

Met dit programma kunt u de C-toonladder spelen door gebruik te maken van de bovenste rij toetsen. Met de regels 10 tot en met 50 wordt een n-lus gevormd, waarmee de toonwaarde van de 8 noten wordt vastgelegd. De instructies 60 en 70 onderzoeken of er een toets wordt ingedrukt. Als dit plaatsvindt zal regel 80 een toontje produceren met een vaste duur en van de juiste toonhoogte. Dit geschiedt door met de "VAL"-instructie de a$-string om te zetten in een getal, waarmee we nadien de n-reeks kunnen registreren.
U kunt dit programma zonder meer uitbreiden met meer toetsen voor het vastleggen van kruis- en molgegevens, maar veel meer kunt u er niet mee doen; voor het groots aanpakken van computermuziek zult u toch naar machinetaal moeten overschakelen; BASIC is gewoon te traag!

**Automatische muziek**

U kunt proberen de computer zelf volledig automatisch muziek te laten componeren. Hoewel dit zeer gecomplideerd is en ver buiten het bestek van dit boek en zelfs van de Spectrum valt, kunt u toch wel wat experimenteren en enige interessante nieuwe mogelijkheden van uw machine ontdekken.
Volledig willekeurige muziek ontstaat door:

```
10 BEEP RND,INT(20-40*RND)
20 GOTO 10
```

Zeer indrukwekkend, maar niet iets om lang naar te luisteren!
De opeenvolging van geluiden moet worden geordend, wil het op muziek gaan lijken.
Een kleine verbetering ontstaat als u er op aandringt dat de computer de toonwaarde van de tonen slechts één eenheid per noot verandert.

```
10 LET p=INT(40-80*RND)
20 BEEP .1,p
30 LET p=p+SGN(.5-RND)
40 GOTO 20
```

Het geheim zit verscholen in regel 30, waar we naar willekeur de waarde van p met één eenheid vermeerderen of verminderen.
Het resultaat klinkt als iets beter, maar gaat na een tijdje toch vervelen. Verder kunt u eigenlijk niet veel meer doen: door de computer gecomponeerde muziek is nog steeds het middelpunt van intensief onderzoek.

**Het oppeppen van het geluid**

Waarschijnlijk bent u ondertussen ontzettend moe geworden van het gepiep uit het kleine luidsprekertje van de Spectrum.
Gelukkig kunnen we het gecomponeerde geluid zonder veel problemen op een normaal geluidsniveau reproduceren. We hebben er niet eens

extra apparatuur voor nodig! Het door de "BEEP"-instructie geproduceerde geluid staat ter beschikking op de "MIC"- en "EAR"-uitgangen op de achterzijde van de machine.
Het enige wat u moet doen, is de cassette-recorder, die u toch reeds gebruikt voor het bewaren en inlezen van programma's in de stand "opnemen" zetten als de Spectrum bezig is met het uitvoeren van geluidsprogramma's. Alle geluiden worden dan op de band opgenomen en kunnen na het terugspoelen van de band op ieder gewenst geluidsniveau worden afgespeeld. Daarnaast kunt u, als u dat wilt, uw Spectrum op orkeststerkte beluisteren, door een extra versterker en luidspreker op de "EAR"-uitgang aan te sluiten.
Als u elektro-technische interesses heeft, kunt u de Spectrum ook gebruiken als signaalgenerator. Het signaal op de "EAR"-uitgang is namelijk een blokgolf, waarvan de frequentie wordt gegeven door de volgende formule:

$$f = 2^{p/12} . fc$$

In deze formule staat p voor de toonwaarde en fc voor de frequentie in Hertz van de middelste C-noot, dus 261,6 Hz.
Hoewel dit een vrij gecompliceerde formule is, heeft de Spectrum geen enkele moeite met de uitwerking.
Formules uitrekenen is immers de voornaamste taak van computers!
Wel moeten we het apparaat daarbij helpen, door hem het volgende programma in te fluisteren:

```
10 INPUT "toonwaarde =";p
20 LET f=261,6*2^(P/12)
30 PRINT"frequentie = ";f;" Hz"
40 GOTO 10
```

Als we daarentegen een bepaalde frequentie willen omzetten in de corresponderende toonwaarde, moeten we het volgende programma gebruiken:

```
10 INPUT "frequentie in Hz = ";f
20 LET p=12*LN(f/261.6)/LN(2)
30 PRINT "toonwaarde = ";p
40 GOTO 10
```

**Geluid en beweging**
Zoals reeds gezegd, is het onmogelijk gelijktijdig geluid te produceren en iets anders te doen. U kunt dus geen bewegende beelden laten begeleiden door geluiden. De Spectrum laat ofwel iets op het scherm bewegen, of houdt zich bezig met het produceren van geluiden; beiden tegelijk kan hij niet aan.
Wat u wél kunt doen, is afwisselend iets laten bewegen en een geluidje opwekken. Uiteraard is dat allesbehalve bevredigend, maar met BASIC kunnen we niets beters bereiken.
Als we bijvoorbeeld een raket willen lanceren met het daarbij horende lawaai, kunnen we als beste resultaat het volgende krijgen:

```
10 LET y=21
20 PRINT AT y,15;"^";
30 BEEP .001,50-y*2
40 LET y=y-1
50 IF y=0 THEN STOP
60 PRINT AT y+1,15;" "
70 GOTO 20
```

Met de regels 20 en 60 laten we een klein projectiel (voorgesteld door een naar boven gerichte pijl) van de onderzijde naar de bovenzijde van het scherm vliegen.
Regel 30 produceert door middel van een "BEEP"-instructie een geluidje na iedere beweging van het projectiel. Omdat ook de toonhoogte van het geluid stijgt als het projectiel de hoogte in schiet, lijkt het alsof er een rechtstreeks verband bestaat tussen de beweging van de pijl en het geluid.
Deze methode kan met tamelijk groot succes worden gebruikt, zolang we er maar voor zorgen dat de bewegingen elkaar vrij snel opvolgen. Is dat niet het geval, dan horen we alleen maar een aantal losstaande geluiden, die niets te maken hebben met de bewegingen die we op het scherm zien.
Anders is het voor het geluid dat we nodig hebben voor het begeleiden van een op de grond kaatsende bal. Hier moeten we het geluid zo kort mogelijk maken. Zolang het geluid duurt, kan de bal immers niet bewegen en een bal die bij het terugkaatsen even stil blijft liggen, maakt een onnatuurlijke indruk!
Gelukkig dat in dit geval zelfs een heel erg kort geluidje een opmerkelijke verbetering van het effect tot gevolg heeft.
Kijk maar naar het verschil bij het "squash"-programma van hoofdstuk 6, als u dat mét en zonder geluid speelt!
Vaak probeert men zoveel mogelijk geluid in een programma te verwerken. Meestal ontstaan dan programma's, die na een tijdje vreselijk gaan vermoeien en zelfs irriteren.

Gebruik alleen geluid als het echt iets toevoegt aan de waarde of de spanning van een programma! Gebruik geluiden om actie te benadrukken, zoals het indrukken van een toets, het kaatsen van een bal of het raken van een doel.
Een minder speels maar zeer nuttig gebruik van geluid is het attenderen van de computer-gebruiker op bijvoorbeeld fouten op het scherm. Een overdaad aan geluid schaadt meer dan het baat!

**Geluidseffecten**
BASIC biedt niet erg veel mogelijkheden om met de Spectrum overtuigende geluidseffecten op te wekken. Wél bestaat er een manier waarmee we de beweging van de conus van de luidspreker rechtstreeks door middel van BASIC-instructies kunnen beïnvloeden.
Externe apparatuur, zoals de ZX-printer, wordt door middel van twee bevelen, "IN" en "OUT" gestuurd door de computer.

Om diverse externe apparaten van elkaar te kunnen onderscheiden, hebben ze een nummer gekregen: hun adres.
De instructie:

```
IN adres
```

leest informatie van het apparaat met het betreffende adres in de computer.
De instructie:

```
OUT adres,waarde
```

stuurt informatie naar het adres van het op de computer aangesloten apparaat.
De "IN"- en "OUT"-instructies behoren eveneens tot de BASIC-instructieset en kunnen worden gebruikt voor het sturen van op de computer aangesloten toestellen, als u hun adres weet. De op de Spectrum aangesloten luidspreker wordt beschouwd als een extern apparaat en heeft adres 254. Deze wetenschap kunnen we in een "OUT"-instructie verwerken voor het opwekken van een klik. Wat de zaak wel een beetje onoverzichtelijk maakt, is het feit dat de kleur van de rand rond het scherm door de computer ook als een extern apparaat wordt behandeld en dat deze kleur bovendien hetzelfde adres heeft als de luidspreker!

De computer raakt dus behoorlijk in de war, wat blijkt uit onderstaand programma:

```
10 FOR i=0 TO 255
20 OUT 254,i
30 NEXT i
```

Soms zal de kleur van de rand veranderen en soms zult u een geluidje uit de luidspreker horen. Dat is verklaarbaar door het feit dat de waarde van de instructie bepaalt welk apparaat, de luidspreker of de randkleur, door het "OUT"-bevel zal worden bestuurd. Zo zullen alle getallen tussen 0 en 15 alleen maar de kleur van de rand om het scherm beïnvloeden. Verklaard vanuit de gedachtenwereld van de computer, kunnen we zeggen dat de eerste drie bits van de waarde de kleur van de rand vastleggen en het vierde bit de luidspreker stuurt. Daarnaast is het belangrijk om te weten dat de computer de informatie over de kleur van de rand bewaart in geheugenlocatie nummer 23624 en wel met een getal dat gelijk is aan acht keer het kleurcodegetal van de kleur van de rand.
Dankzij deze wetenschap kunnen we een programma opstellen, waarmee we de luidspreker kunnen sturen zonder dat de computer in de war raakt met de kleur van de rand rond zijn scherm:

```
10 LET a=PEEK 23624/8
20 OUT 254,a-16
30 OUT 254,a
```

De eerste regel leest de in het geheugen opgeslagen kleurcode van de rand en deelt deze waarde door acht (de "PEEK-instructie komt uitvoerig aan bod in hoofdstuk 7). Het eerste "OUT"-bevel in regel 20 verandert het luidsprekersignaal, het tweede "OUT"-bevel van regel 30 heft deze verandering weer op. Door deze twee tegengestelde stuursignalen zal de luidspreker telkens als het programma wordt gedraaid een klik produceren. Als de lus, regel 40 "GOTO 20" wordt ingevoerd dan doorloopt de computer automatisch het programma en dit veroorzaakt een groot aantal klikken snel achter elkaar, dat we waarnemen als een ratelend geluid.
Het karakter van dit geluid is te beïnvloeden door extra instructies in het basis-programma op te nemen, waardoor er tussen de regels 20 en 30 een vertraging ontstaat. Dat gaat het gemakkelijkst door een variërend aantal "REM"-

instructies tussen te voegen. Na enig geëxperimenteer zult u de machine het geluid van een machinegeweer horen nabootsen.
Een andere categorie geluidseffecten ontstaat door het combineren van een klik met een ander geluid. Zo zal een klik, gevolgd door een hoge toon, ongeveer klinken alsof iets wordt geraakt. Geluiden met zeer lage frequentie worden gebruikt voor het nabootsen van explosies.
Als u gaat experimenteren met de ''OUT''- en ''BEEP''-instructies, zult u ontdekken dat er heel wat leuke geluidseffecten mogelijk zijn.
Zoals reeds vaker in dit hoofdstuk is gezegd, zult u tóch moeten overschakelen op machinecode, als u de geluidsmogelijkheden van uw computer tot op de bodem wil exploiteren.

In principe passen we dan dezelfde technieken toe, maar omdat de computer veel sneller kan werken met machinecode-instructies, zal het maximale aantal klikken per seconde en dus de frequentie van het geluid veel groter zijn.

# 6. BEWEGENDE BEELDEN

Een van de dankbaarste terreinen waarvoor we de computer kunnen gebruiken, is dat van bewegende beelden, ook wel dynamische grafiek genoemd. Hoewel het niet zo voor de hand ligt hoe we van het weergeven van een enkel puntje op het beeldscherm de stap kunnen zetten naar bewegende beelden, zijn de toe te passen technieken vrij eenvoudig.

**Van knipperen naar bewegen**

Als u eerst een enkel punt op het scherm laat schrijven en nadien een "INVERSE 1"-instructie geeft, zal het resultaat een knipperend lichtpuntje zijn:

```
10 PLOT 100,50
20 PLOT INVERSE 1,100,50
30 GOTO 10
```

Het puntje is zo klein, dat het nauwelijks opvalt, hoewel het er echt wel is, ongeveer in het midden van het scherm. Hetzelfde effect, maar veel duidelijker zichtbaar ontstaat als we een blokje [^8] op het scherm laten schrijven, gevolgd door een blanco symbool [8] op dezelfde plaats.
Probeer het maar:

```
10 PRINT AT 10,10;"[^8]";
20 PRINT AT 10,10;"[8]";
30 GOTO 10
```

Het door dit "PRINT"-bevel geproduceerd knipperend vierkantje is acht keer groter dan de punt, geschreven door "PLOT". Het knipperen gaat bovendien sneller, omdat de Spectrum heel wat minder rekenwerk heeft te doen bij een "PRINT"- dan bij een "PLOT"-opdracht.
In beide gevallen kan het knipperen worden vertraagd door het invoeren van vertragingslussen (delay-loops). Lussen, want zou u alleen maar een "FOR"-instructie opnemen, in bijvoorbeeld regel 15, dan zou de zichtbare tijdsduur van het knipperen wél, maar de gedoofde cyclus-duur niet worden verlengd. Om dit laatste te bereiken, moet er ook een "FOR"-instructie worden opgenomen, zeg op regel 25.

U kunt als voorbeeld het volgende programma intoetsen:

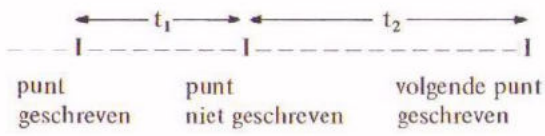
```
10 INPUT ontime
20 INPUT offtime
30 PRINT AT 10,10;"[^8]";
40 FOR i=1 TO ontime
50 NEXT i
60 PRINT AT 10,10;"[8]";
70 FOR i=1 TO offtime
80 NEXT i
90 GOTO 30
```

Meer dan het voorgaande valt er niet te vertellen over hoe u tekst en/of symbolen kunt laten knipperen. Maar nu beweging!
De stap van knipperen naar bewegen is zeer voor de hand liggend. Als u eerst een punt laat oplichten, nadien het punt laat doven en dan het punt daarnaast laat op lichten, lijkt het alsof dat punt één schermpositie is verplaatst. Als u dit proces continu herhaalt, dan lijkt het of één punt zich over het scherm verplaatst. Laat ons als voorbeeld het programma gaan uitwerken, waarmee we een punt in een rechte lijn van de ene kant naar de andere kant van het scherm laten "wandelen". Uit het vorige hoofdstuk weten we hoe we de computer een lijn op het scherm kunnen laten schrijven. Laten we echter met een iets simpeler methode starten!
Als we het punt volgens een horizontale lijn willen laten bewegen, dan hoeven we alleen maar de x-coördinaat te veranderen voor ieder "PLOT"-bevel.
Dit gebeurt als volgt:

```
10 LET y=50
20 FOR x=0 TO 255
30 PLOT x,y
40 PLOT INVERSE 1,x,y
50 NEXT x
```

Dit programma doet precies wat de bewegingstheorie voorschrijft. Regel 30 schrijft een punt op het scherm, regel 40 wist hem weer. Nadien komt regel 30 aan de beurt, die de volgende punt neerschrijft, en zo verder! Als u deze methode gebruikt, moet u er wel zeker van zijn dat alles op het juiste moment plaatsvindt, als u er tenminste prijs op stelt dat de beweging ver-

loopt met de door u gewenste snelheid. In het bovenstaande eenvoudige voorbeeld zijn er twee momenten, die van belang zijn, namelijk het tijdsverloop tussen het schrijven en weer wissen van een punt en het interval tussen het wissen van de oude punt en het schrijven van een nieuwe. Grafisch kunnen we dat als volgt toelichten:




Gedurende het tijdsinterval t1 wordt er wél een punt op het scherm geschreven, gedurende het tijdsverloop t2 is het scherm donker. Het totale interval t1 en t2 is de tijd die nodig is voor een basis-beweging en deze totale tijd bepaalt hoe snel een punt zich over het scherm voortbeweegt. Wat men er meestal niet bij vertelt is hoe men t1 en t2 moet kiezen voor het verkrijgen van een vloeiende beweging. Het antwoord op die vraag is niet zo eenvoudig en in de praktijk gaat men dan ook meestal experimenteren met beide tijden.
In theorie is het allemaal erg eenvoudig. Als u bijvoorbeeld een punt zou zien bewegen achter een geperforeerde metalen plaat, dan zou de tijd dat u die punt kunt zien, overeenkomen met het tijdsinterval t1 en de tijd dat u die punt niet kunt zien, komt dan overeen met het interval t2. Als er erg veel gaatjes volgens een regelmatig patroon in de plaats zijn geponsd, dan ontstaat de indruk dat we de punt ook door de plaat "zien bewegen", omdat onze hersenen een patroon van snel op elkaar volgende gebeurtenissen als een beweging interpreteren.
Met het programmeren van een knipperende en bewegende punt proberen we het effect van een punt achter een geperforeerde plaat na te bootsen, er op rekenend dat onze hersenen dit als een vloeiende beweging zullen interpreteren. Hoe goed we op deze manier een vloeiende beweging kunnen tekenen, is afhankelijk van hoe goed we het effect van "door een geperforeerde plaat kijken", kunnen nabootsen. Als de gaatjes in de plaat dicht op elkaar staan, dan zal de punt voor het grootste deel van de tijd te zien zijn en slechts voor een klein deel van de tijd verborgen zitten achter de plaat. Dan zal het interval t1 veel groter zijn dan het interval t2.
We kunnen stellen dat de voorgaande zin kort en krachtig de eis voor een vloeiende beweging weergeeft. Aan deze eis kunnen we echter met de Spectrum niet zo eenvoudig tegemoet komen. De Spectrum heeft even veel tijd nodig om een punt te schrijven als om een punt te wissen. In principe zijn t1 en t2 dus aan elkaar gelijk.
Maar in de praktijk klopt daar niets van. Alvorens de computer een nieuw punt op het scherm kan schrijven, moet hij enige berekeningen uitvoeren, en dat kost tijd.
Vandaar dat in werkelijkheid het interval t2 veel langer duurt dan het interval t1. Het nadeel van deze on-balans tussen aan- en uit-tijd is dat het bewegend puntje schokkend over het scherm beweegt.
We zouden het programma kunnen aanpassen, door eerst een punt te schrijven, daarna de coördinaten van het volgende punt te laten berekenen en dan pas de oude punt te wissen om vervolgens het nieuwe punt te schrijven. We zouden dan wel de coördinaten van zowel het oude als van het nieuwe punt moeten opslaan in een geheugen, en dat maakt deze aanpassing er niet eenvoudiger op. Een voorbeeldje van zo'n werkwijze geeft:

```
10 LET y=50
20 FOR x=0 TO 255
30 PLOT INVERSE 1;x,y
40 PLOT x+1,y
50 NEXT x
```

Een veel eenvoudiger systeem om een vloeiende beweging te kunnen tekenen, is het invoeren van een tijdverbruikende, maar verder volkomen nutteloze instructie, zoals "35 LET y = y", tussen de schrijf- en wisinstructies. Deze methode is uitermate geschikt om toe te passen bij een machine als de Spectrum, maar natuurlijk staat niets u in de weg om alle besproken manieren uit te proberen. Als we even teruggaan naar het voorbeeld van de geperforeerde metalen plaat, kunnen we nog een belangrijk gegeven opsporen. Als een punt achter de plaat met een snelheid beweegt en telkens voor een tijdsduur van t2 sekonden onzichtbaar is, dan kunnen we berekenen dat de afstand tussen de gaatjes gelijk is aan v * t2.
Hieruit kunnen we afleiden dat het typische

Spectrumprobleem, namelijk een te groot interval t2, ook is op te lossen door de onderlinge afstand tussen de punten te vergroten!
Dat kunnen we programmeren door eerst een punt te schrijven, dan die punt te wissen en in plaats van een nieuwe punt op de eerstvolgende schermlocatie te schrijven er enige ongebruikte punten tussen te laten.
Probeer het volgende programma uit:

```
10 LET y=10
20 FOR i=1 TO 10
30 FOR x=1 TO 31 STEP i
40 PRINT AT y,x;"[^8]";
50 PAUSE 5
60 PRINT AT y,x;"[8]";
70 NEXT x
80 NEXT i
```

Dit programma zal een vierkantje van links naar rechts over het scherm laten bewegen, door de lus in de regels 30 tot en met 70. De eerste keer is de afstand tussen de getekende vierkantjes 1 lokatie, de tweede keer 2 lokaties en zo verder tot er telkens 10 lokaties ongebruikt blijven. Het interessante van dit programma is, dat de indruk van beweging blijft bestaan, zelfs als we 6 of 7 plaatsen overslaan!
De gelijkmatigste beweging ontstaat bij een stap van twee á drie lokaties.
Hoewel deze methode interessante toepassingen heeft, kunnen we hem niet altijd gebruiken. Sommige dynamische programma's eisen bewegingen die per schermpunt gaan.

**Kaatsende ballen en snelheid**

Nu we de basis-systemen onder de knie hebben voor het laten bewegen van een punt over het scherm, kunnen we deze technieken leren toepassen in echte programma's. De beweging kan ook anders dan rechtlijnig zijn, door bijvoorbeeld alle punten op de omtrek van een cirkel of een vierkant één voor één te schrijven en weer te wissen. Laten we ons nu gaan bezighouden met het simuleren van enige praktisch bruikbare bewegingen.
Vaak zullen we in spelletjes behoefte hebben aan een programma, waarmee we de beweging van een bal kunnen nabootsen. Dit kunnen we het eenvoudigst simuleren, door twee snelheden voor de balbeweging te definiëren. Een bal kan zich immers in alle mogelijke richtingen bewegen en iedere balpunt op het scherm is een aantal vertikale en een aantal horizontale lokaties verwijderd van het vorige punt.

Omdat iedere stap hetzelfde tijdsinterval kost (t1 + t2), kunnen we de vertikale beweging de "vertikale snelheid" en de horizontale beweging de "horizontale snelheid" noemen. Bij het schrijven van een nieuw punt volstaat het om de horizontale en vertikale snelheden op te tellen bij respektievelijk de x- en de y-coördinaten van het oude punt.
Probeer dit uit aan de hand van het volgende programma.

```
10 LET v=1
20 LET h=1
30 LET x=0
40 LET y=0
50 PRINT AT y,x;"[^8]"
60 PRINT AT y,x;"[8]"
70 LET x=x+h
80 LET y=y+v
90 GOTO 50
```

Door dit programma zal de bal van de linker bovenhoek naar de rechter onderhoek van het scherm bewegen. Omdat de bal "van het scherm valt", zal dit programma eindigen met een foutmelding. Het ligt voor de hand, dat we de bal moeten laten terugkaatsen tegen de randen van het scherm.
Hoe doen we dat?
Omdat we zo verstandig zijn geweest de beweging van de bal te ontleden in een vertikale en in een horizontale snelheid, is het antwoord op die vraag verrassend eenvoudig. Als de bal een vertikale muur tegenkomt, bijvoorbeeld de rechterrand van het scherm, dan mag hij natuurlijk niet doorgaan in dezelfde richting. In feite is de enige manier om te verhinderen dat de bal door de muur gaat, het omkeren van de horizontale snelheid!
Uiteraard wordt de vertikale snelheid niet beïnvloed door zo'n ontmoeting.
Als regel kunnen we dus stellen dat bij een botsing met een vertikale hindernis de horizontale snelheid moet worden geïnverteerd en dat bij een botsing met een horizontale hindernis het de beurt is aan de vertikale snelheid om te worden omgekeerd.
Als we deze twee regels in een programma verwerken, ontstaat:

```
10 LET v=1
20 LET h=1
30 LET x=0
40 LET y=0
50 PRINT AT y,x;"[^8]"
60 PRINT AT y,x;"[8]"
70 LET x=x+h
```

```
80 LET y=y+v
90 IF x=0 OR x=31 THEN LET h=-h
100 IF y=0 OR y=21 THEN LET v=-v
110 GOTO 50
```

Zonder meer een zeer eenvoudig programma, voor wat u er voor terugkrijgt!
De regels 90 en 100 attenderen de machine op de aanwezigheid van een horizontale of een vertikale hindernis. Als een hindernis present is, wordt de bijbehorende snelheid omgekeerd wat, zo u dat nog niet in de gaten had, gewoon gebeurt door het plaatsen van een -teken voor de waarden van de snelheid.
Omdat bij de "PRINT AT"- en de "PLOT"-instructies de schermplaatsen op een verschillende manier worden genummerd, zal er bij iedere instructie een verband bestaan tussen de richting van de beweging en de polariteit van de snelheid. Bij een "PRINT AT" zal een positieve vertikale snelheid een beweging van boven naar beneden veroorzaken, bij de "PLOT" zal dezelfde positieve snelheid objekten van beneden naar boven sturen.
Hoewel dit programma al tamelijk indrukwekkende bewegende beelden oplevert, kan het toch nog worden verbeterd! Op de eerste plaats schokt de bewegende bal meer dan noodzakelijk is. De tijd tussen het schrijven en weer wissen van een punt kan worden vergroot, door regel 50 te verplaatsen en er een nieuwe regel 85 van te maken (vergeet regel 110 niet te veranderen in "GOTO 60"!)
Als u dit programma laat uitvoeren, zult u merken, dat de beweging al veel vloeiender wordt.
Een tweede verfijning is het invoeren van een "BEEP"-bevel na iedere terugkaatsing.
We moeten de regels 90 en 100 op de volgende manier wijzigen:

```
90 IF x=0 OR x=31 THEN LET h=h-1:BEEP .01,10
100 IF y=0 OR y=21 THEN LET v=v-1:BEEP .01,10
```

Deze eenvoudige uitbreiding van het programma is zeer zeker de moeite waard, het terugkaats-effect wordt er zeer door versterkt. Wél worden we nu meer bewust van het volgende schoonheidsfoutje. Bij iedere botsing blijft de bal even stilliggen. Zeker als u de moeite neemt de duur van de "BEEP" te verlengen tot 0.1 zult u dit verschijnsel zeer duidelijk opmerken.

De oorzaak hier ligt in het verschil in verwerkingstijd van een programmastap door de computer. De machine moet meer berekeningen maken bij een terugkaatsing dan bij een gewone balverplaatsing!
Het enige dat u daartegen kunt doen, is een vertraging opnemen voor de normale beweging van de bal, zodat iedere bewegingsstap van de bal even veel tijd gaat kosten.
Het uiteindelijke programma wordt:

```
10 LET v=1
20 LET h=1
30 LET x=0
40 LET y=0
60 PRINT AT y,x;"[8]"
70 LET x=x+h
80 LET y=y+v
85 PRINT AT y,x;"[^8]"
90 IF x=0 OR x=31 THEN LET h=h-1
   BEEP .01,10:GOTO 60
100 IF y=0 OR y=21 THEN LET v=v-1
    BEEP .01,10:GOTO 60
105 PAUSE 3
110 GOTO 60
```

De regelnummers van dit programma zijn aangepast aan de regelnummers van de vorige programma's, zodat we ze met elkaar kunnen vergelijken. Het bevel "PAUSE 3" in regel 105 wordt alleen uitgevoerd als de bal niet terugkaatst. De duur van deze pauze is proefondervindelijk zo vastgesteld, dat de beweging van de bal over het scherm zo natuurgetrouw mogelijk overkomt. Door deze vertraging wordt het gehele programma trager, maar ja, perfectie eist zijn tol!

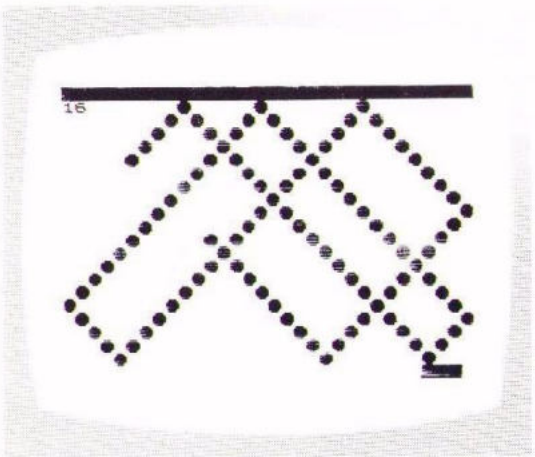
**Squash**

Een zeer uitgewerkt programma, waarin alle technieken van de "kaatsende bal" zijn verwerkt, is het onderstaande voorbeeld van een programma, waarmee men squash kan spelen.

```
10 LET ball=0
20 GOSUB 800
30 INK 0:PAPER 6:CLS
40 LET a=10:LET b=10
50 LET v=1:LET w=1
60 LET x=10:LET y=21
70 LET ball=ball+1
80 GOSUB 500
90 PRINT ball
100 GOSUB 200
110 GOSUB 700
120 GOSUB 300
130 IF b<>21 THEN GOTO 100
140 BEEP .5,-10
150 GOTO 30
200 LET a$=INKEY$
210 IF a$="5" AND x>0 THEN LET x=x-1
220 IF a$="8" AND x<27 THEN LET x=x+1
230 RETURN
300 PRINT AT b,a;"[8]";
```

```
310 LET a=a+v
320 LET b=b+w
330 IF a=31 OR a=0 THEN LET v=-v:BEEP .01,10
340 IF b=1 THEN LET w=-w:BEEP .01,10
350 IF b+1=y THEN GOSUB 400
360 PRINT AT b,a;"[a]";
370 RETURN

400 LET r=a
410 IF r<1 OR r>3 THEN RETURN
420 LET w=-w
430 BEEP .01,0
440 RETURN

500 FOR i=0 TO 31
510 PRINT AT 0,i;"[^8]";
520 NEXT i
530 RETURN

700 PRINT AT y,x;"[8][^8][^8][^8][8]";
710 RETURN

800 POKE USR "a"+0,BIN 00111100
810 POKE USR "a"+1,BIN 01111110
820 POKE USR "a"+2,BIN 11111111
830 POKE USR "a"+3,BIN 11111111
840 POKE USR "a"+4,BIN 11111111
850 POKE USR "a"+5,BIN 11111111
860 POKE USR "a"+6,BIN 01111110
870 POKE USR "a"+7,BIN 00111100
880 RETURN
```

Het programma start met het definiëren van de vorm van de bal door middel van subroutine 800. Daarna wordt met subroutine 500 de muur op het scherm getekend. Subroutine 300 tekent de bal met coördinaten "a, b" op het scherm, daarbij rekening houdende met de horizontale snelheid "v" en de vertikale snelheid "w". Als de bal tegen de muur botst of de randen van het "veld" raakt, wordt de snelheid geïnverteerd, zodat de bal terugkaatst.
Subroutine 700 is verantwoordelijk voor het tekenen van de plaats van het slaghout op de lokatie "x, y". Het is niet noodzakelijk een wisbevel voor de oude positie van het slaghout op te nemen, vanwege de twee blanco grafische symbolen aan het begin en het einde van het slaghout. De twee enige te bespreken nieuwigheden in dit programma zijn; de manier waarop het slaghout wordt bewogen en de manier waarop de bal terugkaatst tegen het slaghout. Om het hout te bewegen, hoeven we alleen maar zijn horizontale coördinaat te wijzigen en wel afhankelijk van welke toets wordt ingedrukt. Wordt de "linker-pijl"-toets ingedrukt, dan wordt er één eenheid afgetrokken van de x-coördinaat. De slaghoutpositie verschuift een plaats naar links. Als men de "rechterpijl"-toets indrukt, wordt de waarde van x met één eenheid verhoogd, waardoor het beeld een plaats naar rechts opschuift. Dit alles staat onder controle van subroutine 200.




Alvorens de nieuwe stand van het slaghout wordt bijgewerkt, kijkt de computer of het symbool nog volledig binnen het scherm valt.
Het terugkaatsen van de bal op het slaghout wordt uitgewerkt door subroutine 400. Als de bal en het hout zich op dezelfde vertikale positie bevinden, schakelt regel 350 subroutine 400, in die controleert of de bal ook de juiste horizontale positie heeft om te worden teruggekaatst. Als dat het geval is, wordt de vertikale snelheid geïnverteerd. Als de bal door het slaghout wordt gemist en van het scherm wil verdwijnen (gedetecteerd door regel 130), dan stuurt een "GOTO" de computer naar regel 100 en een nieuw spel wordt gestart.
Het spel wordt gespeeld door de commando's op de regels 100 tot en met 120 steeds opnieuw te doorlopen. Regel 100 roept de subroutine op die het bewegen van het slaghout controleert, regel 110 zet het slaghout op de juiste plaats op het scherm en regel 120 verwijst naar de subroutine, die de beweging van de bal en het terugkaatsen onder zijn hoede heeft.

**Vrije beweging en zwaartekracht**

In de vorige paragrafen hebben we het bewegen van een bal in een afgebakende ruimte behandeld en gezien hoe we de bal tegen de wanden van de ruimte konden laten terugkaatsen.
Er is echter een heel ander bewegingspatroon mogelijk voor een bal: hij kan in de lucht worden gegooid!
Laat ons pogen een computersimulatie op te stellen voor de beweging van een bal door de

lucht, onderworpen aan de wet van de zwaartekracht.
In de ruimte, waar geen zwaartekracht heerst, zal een bal die eenmaal een zetje in een bepaalde richting heeft gehad, voor eeuwig in die richting en met dezelfde snelheid blijven voortijlen!
Dat geldt natuurlijk niet als bij per ongeluk in botsing zou komen met een ander voorwerp: hij zou dan worden teruggekaatst en met dezelfde snelheid zijn weg in een nieuwe richting vervolgen.
Wat we tot nu toe hebben geleerd over beweging, komt dus overeen met de beweging van voorwerpen in de ruimte, waar we geen rekening hoeven te houden met de invloed van de zwaartekracht.
Een programma dat een balworp zonder invloed van de zwaartekracht simuleert, is:

```
10 LET v=0
20 LET h=1
30 LET x=0
40 LET y=0
50 PRINT AT y,x;"[^B]"
60 PRINT AT y,x;"[B]"
70 LET x=x+h
90 LET y=y+v
100 GOTO 50
```

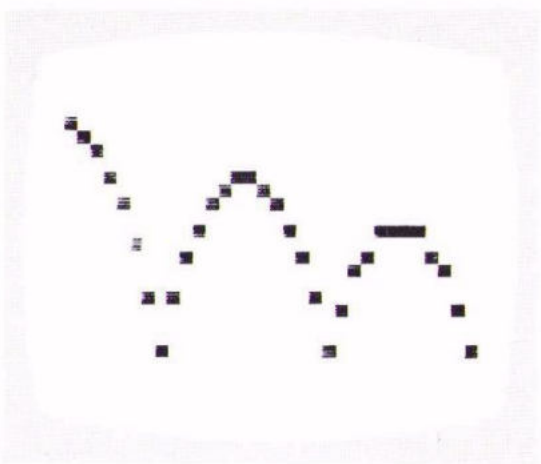
Uit de informatie op de regels 10 en 20 blijkt dat we te maken hebben met een bal, die horizontaal wordt geworpen vanaf het bovenste deel van het scherm. Het lijkt op het van een steile rotswand duwen van een voorwerp. In plaats van naar beneden te vallen, zal het voorwerp gewoon rechtlijnig gaan bewegen, er is immers geen zwaartekracht!
Als we wél rekening houden met de zwaartekracht, zullen we de verticale snelheid moeten variëren. Als we een bal gewoon uit onze handen naar beneden laten vallen, dan zien we dat zijn vertikale snelheid steeds groter wordt, met andere woorden: de bal versnelt.
Voor iedere constante verplaatsing in horizontale richting zal de vertikale snelheid met een vaste waarde toenemen. De grootte van deze vaste waarde hangt af van de grootte van de zwaartekracht, maar voor onze doeleinden kunnen we de waarde het best zo kiezen, dat er een mooi prentje op het scherm wordt getekend.
Om de invloed van de zwaartekracht op het vorige programma te demonstreren, moeten we de volgende regel extra invoeren:

```
80 LET v=v+.1
```

Dit programma bootst een val volgens een parabolische kurve na. Het programma eindigt met een foutmelding, zodra de bal van de onderkant van het scherm "valt".
U kunt dit programma verder verbeteren, door bijvoorbeeld de horizontale snelheid een klein beetje te laten afnemen, waarmee de invloed van de luchtweerstand wordt nagebootst. We kunnen onze kennis over de effecten van de zwaartekracht combineren met wat we weten over terugkaatsende ballen. Als we een horizontale barrière met coördinaat "y = 15" oprichten, dan kunnen we de vallende bal tegen deze barrière laten kaatsen, door de vertikale snelheid te inverteren.
Het programma ziet er dan als volgt uit:

```
10 LET v=0
20 LET h=1
30 LET x=0
40 LET y=0
50 PRINT AT y,x;"[^B]"
60 PRINT AT y,x;"[B]"
70 LET x=x+h
80 LET v=v+.6
90 LET y=y+v
100 IF y>15 THEN LET v=-v
110 GOTO 50
```

Als u regel 6 even wegdenkt, dan ziet u het onderstaand beeld op uw scherm.



U zou nu in staat moeten zijn ballen en andere bewegende voorwerpen volledig op het scherm te controleren en ze die bewegingen te laten uitvoeren, die u in uw hoofd hebt. Het gebruik van de horizontale en vertikale snelheden maakt het opstellen van gelijk welke beweging zeer een-

voudig. Als u de bal wil laten versnellen, hoeft u alleen een kleine waarde bij de overeenkomstige snelheid op te tellen. Als u de bal wil laten vertragen, dan moet u een kleine waarde van de overeenkomstige snelheid aftrekken.

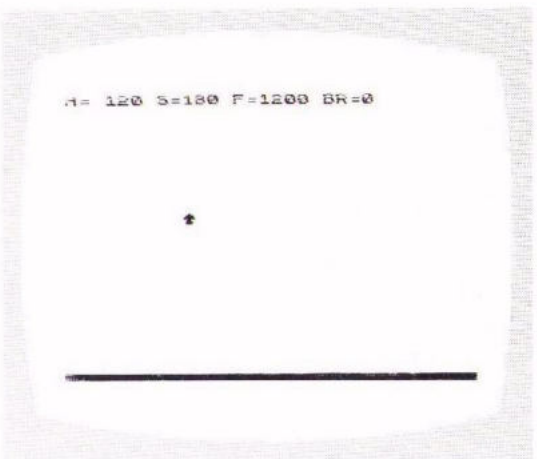
**Maanlander**

Als we enige extra grapjes toevoegen aan het programma van de vallende bal, kunnen we een bevredigend maanlanderspelletje samenstellen. Een op de maan landende raket gedraagt zich namelijk net zo als een vallende bal, met die uitzondering dat de raket motoren kan ontsteken, waardoor zijn vertikale snelheid kan worden verminderd.

```
10 LET f=1200
20 GOSUB 300
30 LET h=0:LET v=0
40 LET h=RND
50 LET x=0
60 LET y=1
70 PRINT AT y,x;"[a]";
80 GOSUB 170
90 PAUSE 3: PRINT AT y,x;"[8]";
100 LET x=x+h
110 IF x>30 THEN GOSUB 250
120 LET y=y+v
130 IF y<19 THEN GOTO 70
140 IF v>.5 THEN PRINT AT 10,10;"**Boem**"
150 IF v<=.5 THEN PRINT AT 10,3;
 "**Spectrum is geland**"
160 STOP

170 LET b$=INKEY$
180 IF b$="" THEN GOTO 200
190 LET b=VAL b$ *10
200 LET f=f-b
210 IF f<0 THEN LET b=0
220 LET v=v-b/1000+.05
230 PRINT AT 0,0;"H= ";INT(11.6*(19-y));
 " S=";INT(200*v);" F=";f;" BR=";b;" "
240 RETURN

250 LET x=0
260 CLS
270 GOTO 380

300 POKE USR "a"+0,BIN 00011000
310 POKE USR "a"+1,BIN 00111100
320 POKE USR "a"+2,BIN 01111110
330 POKE USR "a"+3,BIN 11111111
340 POKE USR "a"+4,BIN 00011000
350 POKE USR "a"+5,BIN 00111100
360 POKE USR "a"+6;BIN 00111100
370 POKE USR "a"+7;BIN 00000000
380 FOR i=0 TO 31
390 PRINT AT 21,i;"[3]";
400 NEXT i
410 RETURN
```

De hoeveelheid brandstof, die u bij de start ter beschikking hebt, wordt vastgesteld in regel 10. U kunt dus het spel minder moeilijk maken door die hoeveelheid brandstof van 1200 eenheden te verhogen.

De raket start met een door het toeval bepaalde horizontale snelheid, vastgesteld in regel 40 en valt onder invloed van de zwaartekracht naar beneden. Tijdens dit proces kunt u de motoren laten werken en een bepaalde hoeveelheid brandstof per tijdseenheid verbruiken, waardoor de valsnelheid daalt. De kracht, waarmee de motoren werken - dus de brandstof die wordt opgebruikt - wordt bepaald door welke van de 0- tot en met 9-toetsen u indrukt. Dit brandstofverbruik wordt op het scherm geschreven als een BR-waarde. Het brandstofverbruik is gelijk aan 10 maal het ingetoetste cijfer. Welke toets u indrukt, wordt gecontroleerd aan de hand van regel 170, door gebruik te maken van de "INKEY$"-instructie. U moet wel de gewenste toets ingedrukt houden, omdat de instructie bij elke lus opnieuw gaat kijken of er nog wel op een toets is ingedrukt.




De verbruikte brandstof wordt afgetrokken van de voorraad en op het scherm genoteerd als een F-waarde. Als u te snel al uw brandstof verbruikt, valt u te pletter op het oppervlak van de maan en verschijnt de boodschap "BOEM" op het scherm.

Het doel van het spel is op de maan te landen met een vertikale snelheid van minder dan 100 meter per seconde. Als de raket door een slecht ingeschat gebruik van de motoren aan de rechterkant uit het beeld valt, dan wordt het scherm schoongewist en start het spel opnieuw met de raket in de linker bovenhoek.

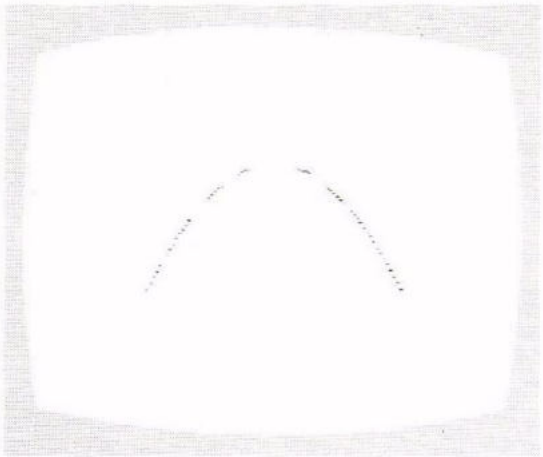
Een veilige landing!

**Gooien in een bepaalde richting**

Tot nu toe hebben we ons bezig gehouden met ballen, die van een bepaalde hoogte al dan niet met een horizontale aanvangssnelheid naar beneden werden gegooid en onder de invloed van de zwaartekracht een bepaalde beweging gingen uitvoeren. Bij veel spelletjes moeten we echter dingen naar boven gooien. Dat kunnen we doen door in een "PRINT"-instructie de y-coördinaten te inverteren of simpelweg door gebruik te maken van een "PLOT"-instructie. Bij "PLOT" vormen de coördinaten "0, 0 " immers het linker onderste punt van het scherm, en bij "PRINT AT" het bovenste linker punt! Probeer het volgende programma uit:

```
10 LET x=0
20 LET y=0
30 LET h=2
40 LET v=4
50 PLOT x,y:PLOT x+1,y
60 PLOT INVERSE 1,x,y:
   PLOT INVERSE 1,x+1,y
70 LET x=x+h

80 LET v=v-.1
90 LET y=y+v
100 IF y<1 THEN STOP
110 GOTO 50
```

De aanvangssnelheid van de bal is horizontaal 2 eenheden en vertikaal 4 eenheden.
Bij iedere stap wordt de vertikale snelheid met 0,1 verminderd, zodat de bal eerst vrij snel de lucht in gaat, daarna vertraagt tot hij zich tenslotte alleen maar horizontaal voorbeweegt. Dan wordt de vertikale snelheid negatief en de bal valt naar beneden met een toenemende snelheid. Omdat we van de "PLOT"-instructie gebruik maken, die punten in plaats van symbolen op het scherm zet, laten we steeds twee punten naast elkaar schrijven, zodat de baan van de bal beter opvalt.
Als we alle geschreven punten in één figuur zouden tekenen, zou de welbekende parabolvorm ontstaan, eigen aan omhooggegooide en nadien weer omlaagvallende voorwerpen.
Meestal willen we de bal onder een bepaalde hoek met een bepaalde kracht omhoog gooien. Deze kracht bepaalt de aanvangssnelheid van de bal; hoe harder u gooit, hoe sneller de bal de lucht in gaat! De werphoek beïnvloedt de verdeling tussen de horizontale en vertikale snelheden. Als u de bal bijvoorbeeld recht omhoog gooit, dan is de vertikale snelheid gelijk aan een bepaalde waarde, maar de horizontale snelheid



gelijk aan nul. Als u de hoek, waaronder u de bal gooit, steeds kleiner maakt, dan neemt de horizontale snelheid toe en de vertikale af.
Als u het probleem zuiver wiskundig zou gaan onderzoeken, dan zou u ontdekken dat, als de aanvangssnelheid "v" is en de werphoek "t", de vertikale snelheid wordt gegeven door de formule" v. sin(t)" en de horizontale snelheid door "v. cos (t)". Deze uitdrukkingen kunnen we zonder problemen opnemen in onze programma's. Wél moeten we er aan denken, dat de Spectrum hoeken niet in graden, maar in radialen meet. Daarom moeten graden worden omgezet in radialen, wat door de volgende omzettingsformule gebeurt:
hoek in radialen = hoek in graden × PI/180.

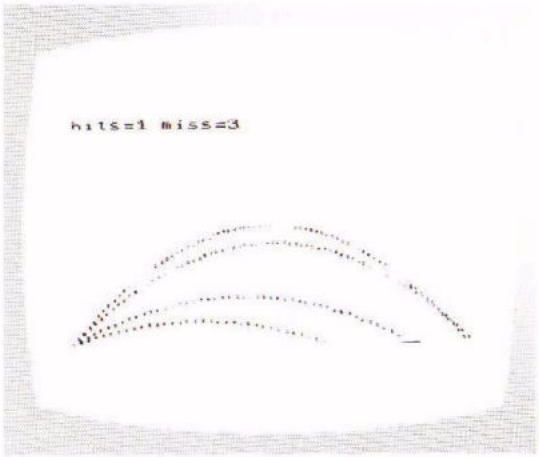
**Een kanonskogel-programma!**

Nu we weten hoe we iets onder een bepaalde hoek en met een bepaalde kracht in de lucht kunnen gooien, gaan we proberen het programma voor een schietspel te ontwerpen.
Als u een kanon aan de linkerkant van het scherm zet en een bepaald doel op een willekeurige plaats aan de rechterkant, dan moet u door middel van twee getallen, namelijk de schiethoek tussen 0 en 90 graden en de kracht van de kanonslading, proberen het doel te raken. In principe is de kracht van de kanonslading onbeperkt, maar waarden rond de 10 voldoen het best; zo heeft de praktijk geleerd. Een extra probleem is dat ieder schot dat de kogel buiten het scherm doet belanden (waar dan ook!) als misser wordt geteld. Dat betekent dat u de schiethoek vrij klein moet kiezen, als u niet

een gat in de bovenzijde van het scherm wil schieten! Op die manier voorkomt u ook, dat u per ongeluk een van uw eigen vliegtuigen neerhaalt. Maar spelenderwijs zal blijken dat richten met laag over de grond scherende kogels moeilijker is, dus deze hoekbeperking is in feite een verrijking van het spel.

```
10 LET b=30
20 GOSUB 300
30 INPUT "kracht=";f
40 LET f=f/2
50 INPUT "Hoek=";t
60 LET h=f*COS(t*PI/180)
70 LET v=f*SIN(t*PI/180)
80 LET x=0
90 LET y=b
100 PLOT x,y:PLOT x,y+1
110 LET v=v-.1
120 PLOT INVERSE 1,x,y:PLOT INVERSE 1,x,y+1
130 LET x=x+h
140 LET y=y+v
150 IF y<=b THEN GOTO 200
160 IF x>255 OR y>150 THEN GOTO 200
170 GOTO 100

200 IF x>=p AND x<=p+10 THEN GOTO 250

210 LET m=m+1
220 GOTO 260

250 BEEP .2,.10:LET s=s+1
260 PRINT AT 1,0;"raak=";s;" mis=";m;"    "
270 GOTO 30

300 LET p=RND*100+100
310 PLOT p,b
320 DRAW 10,0
330 LET m=0
340 LET s=0
350 RETURN
```

Door de subroutine van regel 300 wordt het doel op een willekeurige plaats neergezet met een breedte van 10 punten. Bovendien start deze routine de misser- en treffertellers ''m'' en ''s''. De regels 30 tot 50 voeren de informatie over de hoek en de kracht van de lading in, waarbij de kracht in regel 40 door twee wordt gedeeld, omdat anders de snelheid van de kogel te groot is.




Het middelste deel van het programma is oude koek, namelijk het eerder behandelde programma voor het werpen van een bal, maar uitgebreid met enige aanvullingen waarmee de computer kan oordelen of het doel wordt geraakt. Het is zonder meer mogelijk veel gecompliceerder programma's rond dit thema uit te werken, door bijvoorbeeld rekening te houden met de windrichting en de luchtweerstand. Kortom, er valt voldoende te experimenteren!

**Besluit**

Als u ons verhaal tot nu toe hebt kunnen volgen en alles hebt begrepen, moet u in staat zijn spelletjes zoals de maanlander of het squash-balspel te verbeteren en aan te passen aan uw eigen wensen. U kunt geluiden, mededelingen over het aantal treffers of doelpunten en extra moeilijkheidsgraden gaan uitwerken.
Probeer eens, als voorbeeld, het squash-spel uit te breiden, zodat twee personen kunnen spelen, uiteraard ieder met zijn eigen slaghout!

# 7. PEEK EN POKE

De twee meest mysterieuze instructies uit de BASIC-set zijn zonder enige twijfel ''PEEK'' en ''POKE''. Iedere nieuwe computergebruiker zal zich op een bepaald moment de vraag stellen wat hij (of zij) met deze instructies in de programmeerpraktijk kan doen.
Het antwoord op deze vraag is verre van eenvoudig en bovendien ook nog eens afhankelijk van het merk en het type computer, waarmee u werkt.
In dit hoofdstuk zullen we in het kort de functie van ''PEEK'' en POKE'' gaan uitleggen en enige voorbeelden geven van wat we er op de Spectrum mee kunnen doen. Verwacht echter niet dat u deze kennis op andere computers kunt toepassen, de kans dat het wél gaat is erg klein!

**Wat ''PEEK'' en ''POKE'' doen'**
Van deze twee nieuwe instructies is ''PEEK'' het gemakkelijkst te begrijpen en ook het veiligst in het gebruik.
Een onjuist gebruik van ''PEEK'' kan namelijk onmogelijk een programma verminken. ''POKE'' kan dat wel, let dus goed op bij het toepassen hiervan.
Hoewel we ''PEEK'' tot nu toe een instructie hebben genoemd, is het begrip functie beter op zijn plaats. ''PEEK'' levert namelijk een resultaat op en functies, zoals de berekening van SIN (×), zijn dingen die een resultaat opleveren, meestal een getal als uitkomst van een berekening. Wél is de ''PEEK'' een speciaal soort functie. Het berekent niets maar geeft simpelweg de inhoud van een geheugenplaats als antwoord op een vraag en zet deze inhoud om in een decimaal getal.
Als u bijvoorbeeld:

```
10 LET a=PEEK 7688
```

intoetst, dan zal a gelijk worden aan de inhoud van geheugenplaats 7688. Wilt u de betekenis van zo'n mededeling ten volle begrijpen, dan moet u eerst een heleboel meer weten over computers in het algemeen en over de Spectrum in het bijzonder.

Op de eerste plaats is het van belang te weten, dat computers informatie kunnen opslaan en deze later weer kunnen opzoeken op genummerde geheugenplaatsen. Iedere plaats heeft een exclusief nummer, het adres van de plaats genoemd. Op de tweede plaats moet u weten, dat de hoeveelheid informatie die op een geheugenplaats kan worden ondergebracht, geperkt is. Bij de Spectrum kan iedere geheugenplaats bijvoorbeeld slechts één symbool bevatten. Zoals u wellicht al weet, werkt een computer alleen met enen en nullen en de informatie in het geheugen bestaat bijgevolg ook uit niets anders dan combinaties van enen en nullen. Hoe kan er dan een symbool, zoals een cijfer of een getal, in dat geheugen worden bewaard?
Simpel, iedere combinatie van enen en nullen (in vaktaal noemen we de enen en nullen ''bits'') kan worden omgezet in een getal. Zo stelt de combinatie ''0101'' het cijfer ''5'' voor. Op dit moment is het niet zo belangrijk om te weten hoe die omzetting precies in zijn werk gaat, het is voldoende dat we weten dat het kán. Een combinatie van acht bits, een byte genoemd, kan getallen van 0 (combinatie: 00000000) tot en met 255 (combinatie: 11111111) bevatten. Iedere plaats in het geheugen van de Spectrum kan dus getallen tussen 0 en 255 bewaren. Het wordt nu zo langzamerhand duidelijk, hoe we symbolen als groepen van 8 bits kunnen opslaan!
Op de achterkant van de handleiding van de Spectrum vindt u een lijstje met alle symbolen, die de machine kan opwekken, de zogenoemde ''character-set''. De eerste kolom van dat lijstje, de ''code'', bevat alle getallen van 0 tot en met 255. Daarnaast staan de bijbehorende symbolen.
Een en ander betekent, dat we de inhoud van een geheugenplaats kunnen opvatten als een getal, bijvoorbeeld 76, óf als een symbool CHR$(76), in dit geval ''L''.
De getallen worden in een bepaald onderdeel van de computer, de zogenoemde ''symboolgenerator'', omgezet in de bijbehorende symbolen

en uitgeschreven op het scherm. Het voornaamste dat we van deze uitleg moeten onthouden, is dat iedere geheugenplaats van de computer een getal tussen 0 en 255 kan bevatten. Als u het vorige programma uitbreidt met de instructie "PRINT a", dan zult u zien dat die waar is: de code voor a ligt tussen 0 en 225.
U kunt met hetzelfde programma ook de inhoud van andere geheugenplaatsen bekijken. Dan ziet u dat u getallen krijgt tussen de genoemde grenzen, maar nooit kleiner dan 0 of groter dan 255.

Het enige nog belangrijke punt voor we de funktie "PEEK" in de praktijk gaan brengen, is de vraag hoeveel geheugenplaatsen of met andere woorden adressen er zijn.
Bij de Spectrum zijn de geheugenplaatsen genummerd vanaf 0 tot en met 65535. Niet alle codes hebben een reële betekenis, zoals we later zullen zien, zijn er een heleboel die niet worden gebruikt. Een belangrijk gegeven is verder nog, dat er twee soorten geheugenplaatsen bestaan: RAM en ROM. RAM (Random Access Memory) kunnen we gebruiken voor het opvragen van informatie, maar ook voor het bewaren van informatie. ROM (Read Only Memory) kunnen we alleen maar gebruiken om informatie op te vragen.

Een 16K-Spectrum heeft ongeveer 16000 RAM-geheugenplaatsen maar ook even veel ROM-plaatsen. Deze grote hoeveelheid vast in het geheugen van de machine geprogrammeerde informatie wordt voor zeer vele doeleinden gebruikt, maar één van de voornaamste toepassingen is het vastleggen van de regels van de BASIC-taal.
Het ROM-gedeelte van het geheugen start bij geheugenplaats 0 en gaat tot 16383. Daarna begint het RAM gedeelte, dus bij adres 16384, dat loopt door tot adres 32767 als u een 16K-machine heeft en tot adres 65535 als u de gelukkige eigenaar van een 48K-uitvoering bent.
In een complete 48K-machine zijn alle geheugenplaatsen bezet door ofwel een RAM-, ofwel een ROM-geheugenplaats. Wat we met een "POKE"-instructie kunnen doen, is nu tamelijk eenvoudig te begrijpen. Door middel van een "POKE" kunnen we een byte in gelijk welke RAM-geheugenplaats opbergen. Onzorgvuldig gebruik van deze instructie kan een reeds geschreven programma vernietigen, dus let goed op!
De basis-instructie voor een "POKE" is:

```
POKE address,byte
```

Als u bijvoorbeeld onderstaand programma zonder regelnummers intoetst, want we willen dat de computer het bevel dadelijk opvolgt en we willen verder geen programma in het geheugen opnemen dat misschien toch maar in de weg zit; dan zult u het nummer 33 op het scherm zien verschijnen. Wat is er precies gebeurt? De computer heeft een patroon bits, overeenkomend met het getal 33, op geheugenplaats met adres 17300 opgenomen en als controle deze geheugenplaats met adres 17300 opgenomen en als controle deze geheugeninhoud weer door middel van de "PEEK" op het scherm geschreven.

```
POKE 17300,33
PRINT PEEK 17300
```

Probeer dit programma uit met verschillende gegevens en u zult zien dat het altijd werkt. Behalve natuurlijk, als u een ROM-geheugenplaats uitkiest, maar dat is duidelijk!

**Het tekenen van grote letters met "PEEK"**
Een zeer nuttig gedeelte van het ROM-geheugen is de symboolgenerator. Als u een "PRINT A"-bevel intoetst, moet de computer op de een of andere manier een patroon van inkt- en papierpunten op het scherm tekenen als de hoofdletter A. De computer zoekt het juiste patroon op in een tabel, die wordt bewaard in het ROM-gedeelte van het geheugen. Deze tabel noemt men de symbool-generator en bevat voor ieder symbool dat de Spectrum kan printen het juiste patroon van inkt- en papierpunten.

In hoofdstuk 2, bij de bespreking van de U-D-symbolen, hebben we geleerd hoe een patroon van punten kan worden voorgesteld door hetzelfde patroon maar dan van enen en nullen. Omdat een symbool is opgebouwd uit 8 rijen van ieder 8 punten, kunnen we de vorm van een symbool opslaan in 8 geheugenplaatsen.

Zo kan de letter A als volgt in het geheugen worden bewaard:

| bit-patroon | decimaal code-getal |
|---|---|
| 0 0 0 0 0 0 0 0 | 0 |
| 0 0 1 1 1 1 0 0 | 60 |
| 0 1 0 0 0 0 1 0 | 66 |
| 0 1 0 0 0 0 1 0 | 66 |
| 0 1 1 1 1 1 1 0 | 126 |
| 0 1 0 0 0 0 1 0 | 66 |
| 0 1 0 0 0 0 1 0 | 66 |
| 0 0 0 0 0 0 0 0 | 0 |

Als u de letter A niet in dit patroon herkent, kleur dan alle enen rood. De kolom met decimale getallen komt overeen met wat op het scherm zou verschijnen door de geheugenplaatsen waarin de A zit te "PEEK"-en.
Uit de handleiding van de Spectrum kunnen we afleiden, waar de geheugenplaatsen van de symboolgenerator in de ROM's verscholen zitten. Het adres van het begin van de tabel wordt opgeslagen in de geheugenplaatsen 23606 en 23607. In feite is dit het juiste adres minus 256. Om dus de start van de symboolgenerator te vinden, volstaat het de inhoud van deze twee geheugenplaatsen te "PEEK"-en. We kunnen een groter getal dan 255 in twee geheugenplaatsen bewaren; wel moeten we het dan in twee delen, beiden kleiner dan 255, opsplitsen. In een later stadium zullen we behandelen hoe dat gebeurt. Zodoende kunnen we het nummer reconstrueren door de volgende instructie te gebruiken:

```
PEEK a+256*PEEK(a+1)
```

Hierbij is "a" het adres van de eerste (laagste) geheugenplaats. Uit de handleiding van de machine kunnen we ook afleiden dat de symbolentabel start met het spatieteken en eindigt met het copyright-symbool CHR$ (127).
Rekening houdende met het feit dat ieder symbool 8 geheugenplaatsen nodig heeft, kunnen we de definitie van symbool CHR$ (i) terugvinden in geheugenplaats:

```
start +8*(i-32)
```

waarbij "start" staat voor het begin van de symbolentabel. Nu we kunnen terugvinden waar de symbolentabel zich bevindt, kunnen we de puntpatronen uit deze tabel halen en ze in programma's verwerken, waarmee we bijvoorbeeld grote symbolen op het scherm kunnen schrijven.
Toch moeten we dan eerst een aantal problemen oplossen. Door een "PEEK" kunnen we de codegetallen terugvinden, die in de diverse geheugenplaatsen zijn opgeborgen. We hebben echter niet dat getal, maar het daarmee overeenkomstig patroon van nullen en enen nodig. We moeten dus een manier ontwikkelen, waarmee we een getal kunnen omzetten in zijn eigen patroon van nullen en enen.
Als u iets van wiskunde afweet en het woord "binair getal" u niet vreemd in de oren klinkt, dan is het niet moeilijk om die omzetting uit te voeren. Als u een leek bent op dat gebied, zou het heel wat theorie kosten om u op gelijk niveau te brengen. Dat doen we niet, we gebruiken gewoon het volgende programma, dat een regel met 8 nullen oproept.

```
10 LET start =256*PEEK 23606+256*PEEK 23607
20 LET a=start+8*(65-32)
30 LET r=PEEK a
40 FOR i=7 TO 0 STEP-1
50 LET b=r-2*INT(r/2)
60 LET r=INT(r/2)
70 PRINT AT 20,i;b
80 NEXT i
```

Nu moet u ondertussen voldoende kennis van programmeertechnieken hebben opgedaan, om te zien dat "a" het adres bevat van de startplaats van de 8 bytes die de letter A definiëren. Regel 30 zet het puntenpatroon van de eerste rij van de letter A in de variabele grootheid "r". De "FOR"-lus haalt telkens een bit uit het patroon van "r" en schrijft het resultaat op. De eerste keer wordt de meest linkse bit behandeld, dan de op één na meest linkse, en zo verder tot alle 8 bits op het scherm staan. Omdat we van links naar rechts willen schrijven, gaat de "FOR"-lus van 7 naar 0. De index "i" wordt gebruikt in de "PRINT"-instructie van regel 70. U kunt het volledige patroon voor "r" krijgen, door het programma 8 keer uit te laten voeren, één keer voor iedere rij van het symbool voor de letter A.

```
10 LET start=256*PEEK 23606+256*PEEK 23607
20 LET a=start+8*(65-32)
30 FOR j=0 TO 7
40 LET r=PEEK(a+j)
50 FOR i=7 TO 0 STEP -1
```

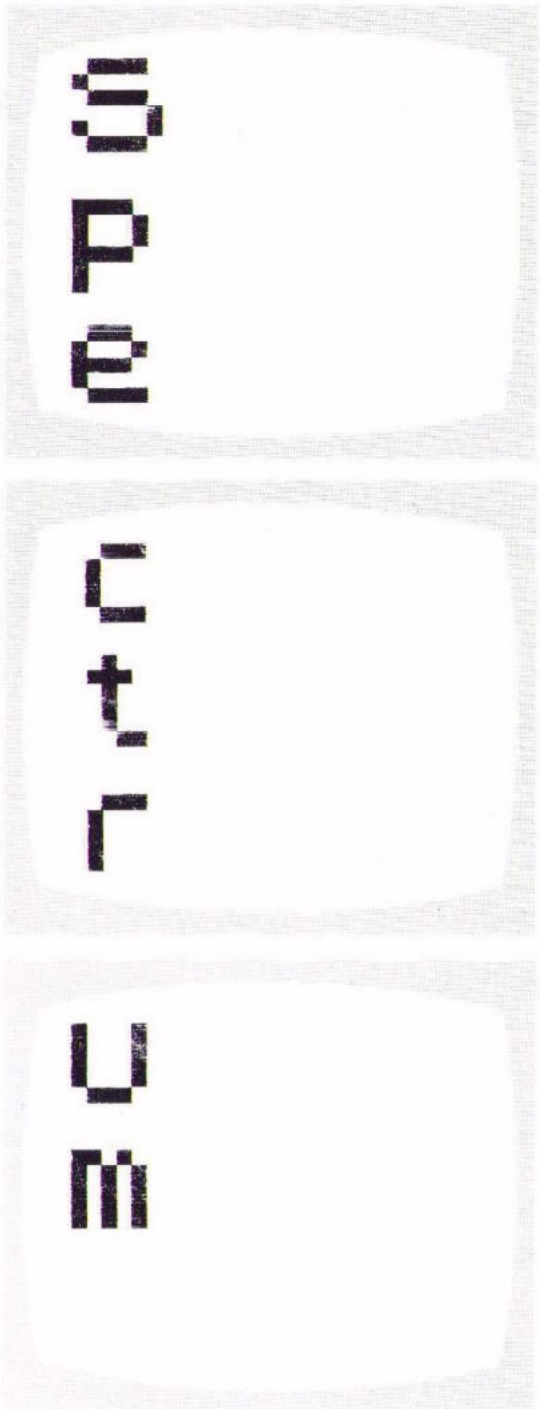
```
60 LET b=r-2*INT(r/2)
70 LET r=INT(r/2)
80 PRINT AT 21,i;b
90 NEXT i
100 PRINT
110 NEXT j
```

Als u de "y" als antwoord intoetst op de vraag "Scroll?" (= verschuiven), zult u het puntenpatroon voor de letter A zien verschijnen. Wij zijn er bijna! We moeten alleen nog enige instructies in het programma verwerken, waarmee we een spatie op het scherm zetten als "b" gelijk is aan 0 en een vierkantje als "b" gelijk is aan 1. Als resultaat verschijnt er dan een grote letter A op het scherm. Voeren wij daarnaast nog wat extra code's in, waarmee we de bijbehorende delen van de tabel van iedere groep van letters verwerken in het programma, dan kunnen we een soort grote lichtkrant op het scherm zetten.
Probeer maar eens:

```
10 INPUT a$
20 LET start=256+PEEK 23606+256*PEEK 23607
30 FOR i=1 TO LEN a$
40 LET a=start+8*(CODE a$(i))-32)
50 FOR j=0 TO 7
60 LET r=PEEK(a+j)
70 FOR k=7 TO 0 STEP -1
80 PRINT AT 21,k;CHR$(128+15*(r-2*INT(r/2)))
90 LET r=INT(r/2)
100 NEXT k
110 PRINT
120 NEXT j
130 NEXT i
```

Iedere boodschap, die u nu op het toetsenbord intikt, zal met opeenvolgende symbolen op de linkerhelft van het scherm verschijnen.

Regel 40 zoekt de positie van iedere letter op in de tabel. De tabel start op het adres wat wordt gegeven door de "start" en het symbool CHR$ (32), dus een spatie, is het eerst aan de beurt. De "CODE"-functie doet het tegengestelde van de "CHR$"-functie. Deze functie zoekt het codegetal op van het symbool dat wordt ingetoetst. Omdat ieder symbool acht geheugenplaatsen omvat, moet de symboolcode met acht worden vermenigvuldigd om op de juiste plaats in de tabel te komen.
In regel 80 wordt de eerder gebruikte techniek toegepast om te bepalen of er een één of een nul ontstaat, maar nu worden er geen enen en nullen geschreven, maar wel CHR$ (128+15*0), dus de spatie in de grafische symbolenset of CHR$ (128+15*1), het gevulde vierkantje ( 8).



Indien u de mededeling wilt laten herhalen, desgewenst voor eeuwig, moet u het programma afsluiten met de regels:

```
130 GOTO 20
140 GOTO 20
```

Met dit programma kunt u grote geschreven letters en symbolen aan gelijk welk spel toevoegen.

**Het ingrijpen in geheugenplaatsen**

Zoals reeds besproken kan één enkele geheugenplaats een getal tussen 0 en 255 bewaren. Het komt echter vaak voor dat we grotere getallen in het geheugen van de computer moeten opbergen. Zo kan, bijvoorbeeld, het adres van een bepaalde geheugenplaats liggen tussen 0 en 65534. Als we om een of andere reden het adres van een geheugenplaats in een ander geheugendeel moeten opslaan, moeten we twee plaatsen gebruiken. Hoe dit in zijn werk gaat, is in theorie gebaseerd op de rekenregels van de binaire rekenwijze, maar hiervan gebruikmaken is een onnodige complicatie voor wie niet ervaren is met deze rekenwijze. Vandaar dat we beter kunnen werken met het manipuleren van gewone decimale getallen, die tòch in de programma-instructies worden gebruikt, en de computer het binaire rekenwerk laten doen.

Omdat een geheugenplaats getallen tot 255 kan bewaren, kunt u met de inhoudscapaciteit van een plaats tellen tot 255. Als u dan een eenheid bij het resultaat zou willen optellen, kunt u het resultaat, namelijk 256, niet in die geheugenplaats kwijt. Wél kunt u die ene eenheid in een tweede geheugenplaats onderbrengen, daarmee aangevend dat u al 255 eenheden hebt geteld. U kunt dan het tellen hervatten in de eerste geheugenplaats (dus vanaf nul, alsof er niets is gebeurd), tot u weer aan het getal 255 toe bent. Kortom, u gebruikt de tweede geheugenplaats om het aantal keren te tellen dat u de eerste hebt ''volgeteld''. De eerste geheugenplaats telt eenheden en wordt daarom de minst belangrijke byte (least significant byte of LSB) genoemd. De tweede geheugenplaats telt veelvouden van 255 en wordt dus de meest belangrijke byte (most significant byte of MSB) genoemd.

Overzichtelijk samengevat:

| eerste geheugenplaats telt eenheden | tweede geheugenplaats telt veelvouden van 255 |
|---|---|
| LSB | MSB |

Het zal nu wel duidelijk zijn, hoe we de inhoud van twee geheugenplaatsen, die samen één adres bevatten, kunnen reconstrueren.

Omdat de eerste plaats eenheden bevat, kunnen we deze plaats gewoon ''PEEK''-en, nadien kunnen we ook de inhoud van de tweede ''PEEK''-en, maar deze inhoud moeten we eerst met 255 vermenigvuldigen en nadien bij de inhoud van de eerste optellen.
Als we dat naar BASIC vertalen, ontstaat:

```
PEEK m+256*PEEK(m+1)
```

Met deze instructie kunnen we de inhoud van de geheugenplaatsen ''m'' en ''m + 1'' zichtbaar maken. Iets in twee geheugencellen plaatsen, dat groter is dan 255 gaat net zo gemakkelijk. U moet het getal dat groter is dan 255 door deze waarde delen en vindt daarmee uit hoe vaak de eenhedenteller ''rond heeft geteld''. Dat resultaat moet u bewaren in de MSB-geheugenplaats, de restwaarde van de deling hoort thuis in de LSB-geheugencel.
In BASIC:

```
POKE m+1,INT(v/256)
POKE m,v-256*INT(v/256)
```

Deze instructie zal het getal ''v'' opslaan in de geheugenplaatsen met de adressen ''m'' en ''m + 1''.
In het voorafgaande hebben we al gebruik gemaakt van de net beschreven methode om twee geheugenplaatsen op hetzelfde tijdstip te ''PEEK''-en. In wat volgt zullen we zien dat het soms ook noodzakelijk is door middel van een gecombineerde ''POKE'' informatie in twee geheugencellen te schrijven!

**Auto Repeat Delay** - Bepaalt hoelang we op een toets moeten drukken, alvorens de machine overschakelt naar automatisch herhalen van het symbool. Deze tijd kunnen we veranderen door de inhoud van geheugenplaats 23561 te ''POKE''-en, de vertraging wordt uitgedrukt in vijftigste delen van een seconde.

**Key Repeat Rate** -Het tempo, waarmee een toets-symbool automatisch wordt herhaald, kan worden gewijzigd door het nieuwe tempo in vijftigste van seconden in geheugencel 23562 te ''POKE''-en.
Deze beide eigenschappen zijn zeer nuttig als we bijvoorbeeld de ''gevoeligheid'' van het toetsen-

bord willen veranderen bij het spelen van spelletjes.

**Keypress Beep** - De tijdsduur van de klik die de luidspreker produceert bij het indrukken van een toets kan worden bepaald door de inhoud van geheugencel 23609 te "POKE"-en. Als u de geheugeninhoud voldoende verhoogt, wordt de klik een echt toontje.

**Start van de U-D symbolen** - In de geheugenplaatsen 23675 en 23676 wordt het startadres van het RAM-geheugendeel vastgelegd, dat wordt gebruikt om de U-D-symbolen te definiëren.

**X- en Y-coördinaten van het laatste getekende punt** - Deze zitten respectievelijk in de cellen 23677 en 23678. Deze twee plaatsen kunnen door een "PEEK" worden uitgelezen als we niet meer weten waar het laatst getekende punt zich bevindt of worden gewijzigd door een "POKE" om het startpunt van een "DRAW"-instructie vast te leggen.

**Scroll-onderdrukking** - In de geheugenplaats met adres 23692 wordt geteld hoe vaak het TV-beeld automatisch een regel naar boven is geschoven, met andere woorden, hoe vaak er is "gescrolled". Om exacter te zijn, de computer bepaalt aan de hand van deze cel wanneer hij "scroll" op het scherm moet zetten.
Soms kan het vervelend zijn, dat de computer deze boodschap op het scherm zet, met name bij het programma voor het schrijven van grote letters. We kunnen dit systeem uitschakelen door af en toe een groot getal, zoals 255, in deze geheugenplaats te zetten door middel van een "POKE". Zo kunnen we deze mededeling in genoemd voorbeeld uitschakelen, door de volgende extra regel op te nemen in het programma:

```
35 POKE 23692,255
```

**Conclusie**
In dit hoofdstuk hebben we een poging gewaagd enig inzicht te geven in de manier waarop de instructies "PEEK" en "POKE" functioneren. Het voorbeeld van het door de computer laten schrijven van grote letters op het scherm is een typische toepassing van "PEEK" en "POKE". Het is niet voldoende alleen BASIC goed te beheersen; dit soort programma's kan onmogelijk worden geschreven als we geen inzicht hebben in hoe de machine is opgebouwd en werkt. We moeten op de hoogte zijn van het bestaan van een symbolengenerator, waar we die kunnen vinden en hoe we er mee om kunnen gaan. Beide soorten kennis zijn essentiëel voor het opstellen van programma's met deze instructies. Als u ooit een andere computer zou gaan gebruiken, dan zal de manier waarop "PEEK" en "POKE" daar werken in wezen wel hetzelfde zijn, maar een Spectrum-programma als het schrijven van grote letters op het scherm zou niet door deze andere computer worden begrepen. Wél kan het programma worden aangepast aan de eigen taal van die computer, door het veranderen van geheugenplaatsen, enzoverder. U moet dan wel op de hoogte zijn van de interne structuur van de machnine.
Op de vraag wat men met "PEEK" en "POKE" kan doen, bestaat geen algemeen antwoord, omdat de toepassingen mede worden bepaald door het type computer dat men gebruikt.

# 8. GEVOEL VOOR TIJD

Iedere computer gaat op een bepaalde manier met tijd om wat door het ontwerp van het apparaat wordt vastgelegd. Sommige ontwerpen bieden de programmeur alle mogelijkheden om in het tijdmechanisme in te grijpen, andere uitvoeringen zijn wat dat betreft zo gesloten als een bus.
De Spectrum zit ergens tussen deze twee uitersten in: hij heeft een echt tijdbevel in zijn instructieset, namelijk "PAUSE", waarmee we toegang krijgen tot de klok van de machine, maar toch zullen we meestal een beroep moeten doen op "PEEK" en soms "POKE", willen we echt gaan ingrijpen in het tijdbesturingssysteem van de machine. Maar laten we eerst bekijken hoe de "hartslag" van de machine wordt opgewekt.

**Een ingebouwde klok**
De in de Spectrum ingebouwde microprocessor, de Z80, is verantwoordelijk voor het opbouwen en in stand houden van het beeld op uw TV-scherm. Een gewoon TV-toestel schrijft iedere vijftigste seconde een beeld op het scherm. De Spectrum moet dus iedere 1/50ste seconde ophouden met wat hij bezig was te doen, om een nieuw beeld op het scherm te schrijven.
De in de computer ingebouwde klok, die onder andere verantwoordelijk is voor het op de juiste momenten schrijven van die 50 TV-beelden per seconde, is een zeer handig hulpmiddel dat in talrijke programma's goed van pas komt. De "PAUSE"-instructie is een van de middelen, waarmee we met de klok kunnen manipuleren. Probeer bijvoorbeeld het volgende programma uit:

```
10 PRINT"tik"
20 PAUSE 50
30 PRINT"tak"
40 PAUSE 50
60 GOTO 10
```

Dit programma zal afwisselend de woorden "tik" en "tak" op het scherm schrijven, met ongeveer een seconde tussenpauze. Hoewel iedere "PAUSE"-instructie een rusttijd van exact een seconde inlast (namelijk de tijd voor het schrijven van 50 beelden) zal de tijd tussen ieder woord toch iets langer zijn, omdat de computer een bepaalde tijd nodig heeft voor het uitwerken van de instructie "PRINT" en "GOTO".

**Vertragings-lussen**
Er zijn tal van computers in de handel, die geen "PAUSE"-instructie hebben en daarom is het nuttig dat u ook een andere manier leert kennen om een bepaalde vaste vertraging in een programma op te nemen: de vertragingslus. Een vertragingslus is een "FOR"-lus, die niets anders doet dan er voor zorgen dat de computer een constante en bekende hoeveelheid tijd verliest.
Een voorbeeld:

```
10 LET t=200
20 PRINT"tik"
30 FOR i=1 TO t
40 NEXT i
50 PRINT"tak"
60 FOR i=1 TO t
70 NEXT i
80 GOTO 10
```

In dit programma worden twee vertragingslussen gebruikt. Iedere lus geeft een vertraging van iets minder dan een seconde, waardoor het interval tussen iedere "tik" en "tak" ongeveer gelijk wordt aan één seconde.
Indien u de nauwkeurigheid precies wilt bepalen, kunt u een heleboel tikken en takken gaan "timen" en berekenen hoe lang het interval is. Als het minder dan een seconde is, kunt u de waarde van 't verhogen; is het interval groter, dan kunt u de waarde van "t" verkleinen.
De tijdsvertraging is verder ook afhankelijk van het soort instructie, dat in de vertragingslus wordt gebruikt. Als u bijvoorbeeld "FOR i= 1 TO t" verandert in "FOR i= 1 TO 200", zal de tijdsvertraging iets variëren.

**De beeldenteller**
De Spectrum kan door gebruik te maken van de "PAUSE"-instructie worden opgedragen gedu-

rende het schrijven van een bepaald aantal TV-beelden niets te doen. Het ligt dus voor de hand om aan te nemen, dat ergens in het geheugen een teller zit, waarmee de machine het aantal op het scherm geschreven beelden telt. Dat is juist; in feite zijn er drie geheugenplaatsen, die voor dit doel worden gebruikt. Deze drie plaatsen gaan door het leven met de naam "FRAME COUNTER" (beeldenteller) en zijn gehuisvest op de adressen 23674, 23673 en 23672. De geheugencel met het laagste adres begint het aantal beelden te tellen op het moment dat de machine wordt ingeschakeld. Maar daar we weten dat een cel slechts 255 beelden kan tellen, is de hoeveelheid beelden die we in deze ene cel met adres 23672 kunnen opslaan zeer beperkt. Vandaar dat cel 23673 het aantal keren telt dat de vorige geheugenplaats een volledige cyclus van 255 beelden heeft verwerkt. De laagste geheugenplaats telt dus beelden; de volgende cel telt veelvouden van 255 beelden. De eerste cel telt van 0 tot 255 voor iedere eenheid die de volgende cel telt.
Op dezelfde manier zal geheugencel 23674 het aantal keren dat cel 23673 is volgeteld registreren. Het systeem is dus te vergelijken met de werking van een traditionele mechanische teller, waar iedere keer als een telwieltje een ronde maakt, het volgende telwieltje een eenheid verder springt.

Samengevat: de eerste cel telt ieder vijftigste deel van een seconde, de tweede cel iedere $256 \times 50$-ste deel van een seconde.
U kunt de tellers aan het werk zien door:

```
10 PRINT PEEK(23672)+256*PEEK(23673)+
   65536*PEEK(23674)
20 GOTO 10
```

Hierin staat het getal 65536 voor het kwadraat van 256! Het verschil tussen de achtereenvolgende getallen komt overeen met het aantal beelden dat de Spectrum op het scherm heeft geschreven tussen twee "PRINT"-cycli. Als we het totale aantal seconden willen laten opschrijven, hoeven we alleen maar het bovenstaande getal te delen door 50. Als we bovendien willen dat de secondeteller, die zo ontstaat, vanaf nul begint te tellen, moeten we door middel van een "POKE"-instructie de drie geheugenlocaties vullen met nul.

Het programma voor de secondeteller is:

```
10 LET p=23672
20 POKE p+2,0
30 POKE p+1,0
40 POKE p,0
50 PRINT PEEK (p)+256* PEEK (p+1)+
   65536* PEEK (p+2)
```

Dit deel van het programma zet de inhoud van de drie tellers op nul. Daarbij is het handig eerst de traagste teller op nul te zetten door middel van de "POKE" en dan pas de snellere tellers, net zoals u bij een gewone klok eerst de uren gelijk zet, dan de minuten en tenslotte de seconden.
Uitbreiding met de volgende regel:

```
60 GOTO 50
```

zet de secondeteller aan het werk.

**Een digitale klok**
We kunnen onze computer op diverse manieren omvormen tot een digitale klok. Een van de eenvoudigste systemen is gebruik te maken van het vorige programma, waardoor de machine het aantal seconden gaat tellen dat is verlopen sinds het inschakelen van de computer. Het enige wat moet gebeuren is het invoeren van de juiste tijd in seconden. Het resultaat zal de computer omzetten in uren, minuten en seconden en deze gegevens op het schema uitschrijven.
Een interessantere manier is gebaseerd op het idee achter het "tik"-tak"-programma. U kunt namelijk de beeldenteller gebruiken om te controleren wanneer er een seconde voorbij is.
Probeer maar eens:

```
10 LET p=23672
20 POKE p,0
30 IF PEEK p<>50 THEN GOTO 30
40 POKE p,50
50 PRINT "tik"
60 GOTO 30
```

Denk nu niet dat er in het bovenstaande programma een drukfout staat, het werkt gewoon niet!
Het interessante aan dit programma is te onderzoeken waarom het niet werkt. In eerste instantie ligt dat niet zo voor de hand, want het idee achter het programma is niet fout. Met regel 20 wordt de laagste beeldenteller op nul gezet. Regel 30 controleert of de inhoud van deze teller

gelijk is aan 50. Is dat niet het geval, dan wordt de regel 30 weer doorlopen en wordt de inhoud van de teller opnieuw met de waarde 50 vergeleken. Is de inhoud van de teller wél gelijk aan 50, dan weten we dat er precies een seconde is verlopen (de teller telt immers 50 beelden per seconde). Terwijl het woord "tik" op het scherm wordt geschreven, wordt de teller opnieuw op nul gezet en start het programma met het aftellen van de volgende seconde. Een goed doordacht programma, waarom werkt het dan niet? De problemen ontstaan, doordat de Spectrum een bepaalde tijd nodig heeft voor het uitvoeren van de BASIC-instructies. Zo kost het meer dan 1/50-ste van een seconde om regel 30 uit te voeren. Met dit bevel kunnen we dus nooit controleren of een teller die telt met 50 pulsen per seconde al dan niet een bepaalde waarde bereikt. Op het moment dat de teller inhoud 50 bereikt, heeft het programma hoogst waarschijnlijk nog niet eens de resultaten van het vorige "IF"-bevel uitgewerkt! Als u dus dit programma vaak achter elkaar laat uitvoeren, zult u vaststellen dat er af en toe een "tik" op het scherm wordt geschreven, namelijk op die toevallige momenten dat het "IF"-bevel de teller uitleest als de inhoud nét vijftig is.
Als u dus de beeldenteller wilt gebruiken als een ingebouwde tijdklok, dan kan dit alleen goed werken als u tijdsintervallen kiest, die lang zijn in verhouding tot de tijd die de computer nodig heeft voor het uitwerken van een "IF"-instructie.
De middelste beeldenteller telt slechts om de 256 beelden, dat is om de 5,12 seconden. Als u geen bezwaar hebt tegen een klok, die niet om de seconde, maar om de 5,12 seconde verder telt, dan kunt u deze beeldenteller gebruiken in het soort programma's waarin we de onderste beeldenteller niet kunnen gebruiken.
Een voorbeeld:

```
10 LET s=0
20 LET h=20
30 LET m=39
40 LET p=23673
50 POKE p,0
60 LET a=PEEK p
70 LET b=a
80 LET a=PEEK p
90 IF a=b THEN GOTO 80
100 LET s=s+5.12
110 IF s<60 THEN GOTO 190
120 LET s=s-60
130 LET m=m+1
140 IF m<60 THEN GOTO 190
150 LET m=0
160 LET h=h+1
170 IF h<24 THEN GOTO 190
180 LET h=0
190 CLS
200 PRINT AT 0,0;h;AT 0,3;m;AT 0,6;INT s
210 GOTO 70
```

Het programma leest de inhoud van de bovenste beeldenteller in regel 60 (a) en nadien opnieuw in regel 80 (b). Als het verschil tussen beide grootheden gelijk wordt aan 1, dan weten we dat er 5,12 seconden voorbij zijn. De secondeteller kan dan opnieuw worden ingelezen met de juiste waarde (regel 100) en de nieuwe tijd op het scherm geschreven door middel van de regels 110 tot en met 200. Uit de eerste drie regels van het programma volgt, dat de klok start op 20 uur en 39 minuten. Ieder gewenst startpunt kan in de regels 10, 20 en 30 worden ingetoetst. Het zal duidelijk zijn, dat we dit programma op verschillende manieren kunnen verbeteren en perfectioneren. U kunt bijvoorbeeld het "grote letter"-programma te hulp roepen om de juiste tijd als een mededeling op het scherm te schrijven of u kunt een alarm- en wekfunctie inbouwen. Het ligt voor de hand de klok echt hoorbaar te laten tikken. In principe is dat mogelijk door gebruik te maken van de "BEEP"-instructie. Denk er echter wel aan, dat een "BEEP" de computer tot werkeloosheid dwingt! Tijdens heet uitvoeren van de "BEEP" zullen dus ook de beeldentellers stilstaan!

**Een schaakklok**

Het onderstaande programma is een zeer eenvoudige toepassing van het gebruik van de beeldenteller in een schaakklok.

```
10 LET tw=0
20 LET tb=0
30 LET g=0
40 LET p=23672
50 PRINT "Druk op 'n toets om spel te starten"
60 IF INKEY$="" THEN GOTO 60
70 CLS
80 POKE p+2,0:POKE p+1,0:POKE p,0
90 LET t=(PEEK(p)+256*PEEK(p+1)+
   65536*PEEK(p+2))/50
100 IF g=1 THEN PRINT AT 5,15;"ZWART ";
    INT((t+tb)/60);".";INT((t+tb)-
    INT((t+tb)/60)*60);" "
110 IF g=0 THEN PRINT AT 5,0;"WIT ";
    INT((t+tw)/60);".";INT((t+tw)-
    INT((t+tw)/60)*60);" "
120 IF INKEY$="" THEN GOTO 90
130 BEEP .1,0
140 IF g=0 THEN LET tw=tw+t
150 IF g=1 THEN LET tb=tb+t
160 LET g=NOT g
170 GOTO 80
```

Er worden geen nieuwe principes in dit programma gebruikt, alles zou dus duidelijk moeten zijn. Wél wordt enige malen teruggegrepen naar technieken die we in de vorige hoofdstukken hebben uitgewerkt.
Het totale tijdsverloop wordt gegeven door "tw" voor wit en "tb" voor zwart. De veranderlijke "g" is 1 als zwart speelt en 0 als wit aan de beurt is. Het indrukken van gelijk welke toets van het toetsenbord schakelt om van wit naar zwart, of omgekeerd, met behulp van regel 120. Het tijdsverloop tussen twee toets-drukken (t) wordt opgeteld bij de totale speeltijd in de regels 100 en 110 voor respektievelijk zwart en wit. De teller wordt nadien teruggezet op nul door regel 80.
Wel moet u er bij gebruik van dit programma op letten, dat een zet niet langer mag duren dan 4 dagen, daar anders de beeldenteller over zijn maximum heen telt en van voren af aan begint!

**Een reaktietijd spelletje**

Door gebruik te maken van de snelste beeldenteller kunnen we gebeurtenissen tot op 1/50-ste seconde nauwkeurig chronometreren. Daarom kunnen we deze teller gebruiken voor het opstellen van een reaktietijdspel. Wél moeten we voor ogen houden dat de nauwkeurigheid van het geheel wordt begrensd door de traagheid waarmee de BASIC-instructies door de machine worden verwerkt en dat dus de theoretische nauwkeurigheid van 1/50-ste seconde in de praktijk nooit zal worden gehaald.
Voor serieuze toepassingen kunnen we het gebruik van dit soort programma's dan ook vergeten, maar voor spelletjes zijn we er dik tevreden mee!

```
10 PRINT "klaar"
20 FOR i=0 TO RND*100+50
30 NEXT i
40 LET p=23672
50 POKE p+1,0
60 POKE p,0
70 PRINT "start"
80 IF INKEY$="" THEN GOTO 80
90 LET t=PEEK(p)+PEEK(p+1)*256
100 PRINT "reaktietijd=";t/50;"sec"
110 GOTO 10
```

Na een willekeurig bepaalde vertraging wordt het woordje "start" op het scherm geschreven. Men moet dan zo snel mogelijk gelijk welke toets indrukken, waarna het tijdsinterval, dus uw reaktietijd, op het scherm verschijnt.
Ook aan dit programma kan heel wat worden gesleuteld om het spannender te maken. Zo zou u de computer het gemiddelde van tien reakties kunnen laten berekenen, het resultaat laten beoordelen en dit met een mededeling als: "te traag", "niet slecht", "goed gedaan!" of "zeer snel" op het scherm laten schrijven.
Probeer het maar eens!

```
10 LET s=0
20 FOR j=1 TO 10
30 CLS
40 PRINT "klaar"
50 FOR i=0 TO RND*50+40
60 NEXT i
70 LET p=23672
80 POKE p+1,0
90 POKE p,0
100 PRINT "start"
110 IF INKEY$="" THEN GOTO 110
120 LET t=PEEK(p)+PEEK(p+1)*256
130 LET t=t/50
140 LET s=s+t
150 NEXT j
160 CLS
170 PRINT"Uw gemiddelde is ";s/1
180 IF s/i>.08 THEN PRINT "te traag"
190 IF s/i>=.05 AND s/i<=.08 THEN PRINT
    "niet slecht"
200 IF s/i<.05 THEN PRINT "goed gedaan"
210 IF s/i<.02 THEN PRINT "zeer snel"
```

De laatste regels van het programma berekenen het gemiddelde van tien reaktietijden en schrijven de toepasselijke beoordeling op het scherm. Dit programma kunnen we als uitgangspunt gebruiken voor een heleboel leuke spelletjes. Denk er echter steeds aan, dat de nauwkeurigheid vrij klein is!

# 9. STRINGS EN WOORDEN

Vaak denkt men dat computers alleen maar in staat zijn nummers en getallen te verwerken en ingewikkelde wiskundige berekeningen uit te voeren. Ondertussen moet u er wel achter zijn gekomen, dat dit niet zo is. Met uw Spectrum kunt u net zo gemakkelijk met letters en woorden omgaan als met cijfers en getallen!
Toch zijn er een aantal problemen op te lossen als we woorden of, meer algemeen, tekst in spelletjes willen gaan verwerken. Sommige van deze problemen zijn vrij eenvoudig op te lossen, andere confronteren ons met de grenzen van onze kennis over computers. Problemen, die te maken hebben met begrippen als kunstmatige intelligentie.

**STRINGS en dat soort zaken**

Alvorens in te gaan op hoe we strings in praktische programma's kunnen gebruiken, zullen we een kort overzicht geven van de manieren, waarop we de Spectrum strings kunnen laten behandelen.
De Spectrum maakt een onderscheid tussen een string en een andere variabele grootheid, door het symbool van de variabele te laten volgen door het teken "$".
Een voorbeeld:

```
10 LET a$="Naam"
```

Een string mag net zo lang zijn als men wil, als hij maar in het geheugen van de machine past. De machine kan op drie manieren met strings werken.
De eerste manier voegt strings samen, door het gebruik van het +-teken. Een voorbeeld:

```
10 LET a$="Voornaam"
20 LET b$="Familienaam"
30 LET a$=a$+b$
40 PRINT a$
```

Dit programma verbindt de reeksen "voornaam" en "familienaam" in de variabele "a$".
Met de tweede methode kunt u gelijk welk deel van een reeks selecteren door gebruik te maken van een "deel-notatie".
Zo zal:

```
10 LET a$="abcdefg"
20 PRINT a$(2 TO 5)
```

de string "abcdefg" van de tweede tot en met de vijfde letter op het scherm schrijven, dus "bcde".
In het algemeen kunt u de instructie [a$ "begin" "TO" einde], waarin natuurlijk "begin" en "einde" worden vervangen door getallen, gebruiken om een a$ aan te duiden, startend bij "begin" en eindigend bij "einde". Met de Spectrum kunnen we de "deel-notaties" als volgt verkorten:

```
a$ (TO n)=a$(1 TO n)
a$ (n) =a$(n TO n) i,e de n-de letter
a$ (n TO )=a$(n TO LEN (a$))
a$(TO)=a$ (1 TO LEN (a$)) i,e dan hele reeks
```

De derde methode om met strings om te gaan is erg slim bedacht. U kunt een deel van een string wijzigen door gebruik te maken van een "deel-notatie".

```
10 LET a$="abcdefg"
20 LET a$(2 TO 3)="12345"
30 PRINT a$
```

zal de string "abcdefg" omzetten in de string "a12defg". De "deel-notatie" bepaalt dus het deel van de oorspronkelijke string dat we willen veranderen, in dit voorbeeld de tweede en derde letter. Het maakt daarbij niets uit of de string na het gelijkheidsteken groter is dan het deel wat we willen veranderen. De computer zal steeds het juiste aantal symbolen van links af aan beginnen te tellen. In het voorbeeld worden alleen "1" en "2" gebruikt, hoewel de string "12345" veel langer is.
U kunt met deze technieken wat praktijkervaring opdoen door middel van het volgende programma:

```
10 INPUT a$
20 LET b$=""
30 FOR I=LEN a$ TO 1 STEP -1
40 LET b$=b$+a$(I)
50 NEXT I
60 PRINT b$
```

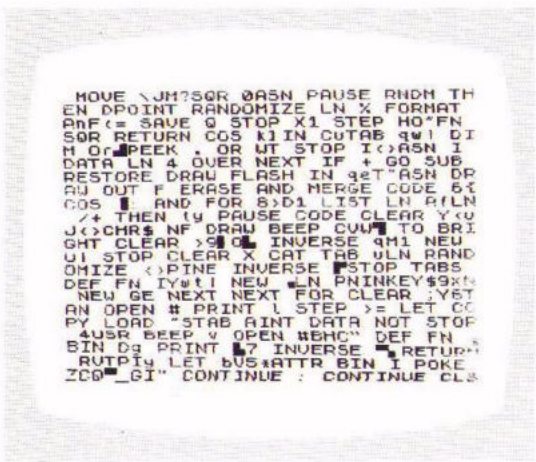
Hiermee kunnen we gelijk welke string inlezen en nadien omkeren. U kunt bijvoorbeeld de volgende mededeling intoetsen: ''tupni ed treetrevni ammargorp tid'' en kijken wat er op het scherm verschijnt.
Het programma ''rafelt'' het geschrevene uiteen in enkele letters en bouwt de tekst daarna weer op in de omgekeerde volgorde door de instructie in regel 40.
In de praktijk kunt u dit programma gebruiken voor het versturen van geheime boodschappen of voor het leren achterste voren te praten!

**Woorden en toeval**

Hoe we door het toeval bepaalde getallen kunnen opwekken, hebben we besproken in hoofdstuk 2. In hoofdstuk 3 hebben we aansluitend het principe behandeld, waarmee we door het toeval bepaalde grafieken kunnen genereren.
Door gebruik te maken van dezelfde technieken, kunnen we ook toevals-letters en -symbolen op het scherm zetten. Toets bijvoorbeeld het volgende programmaatje in:

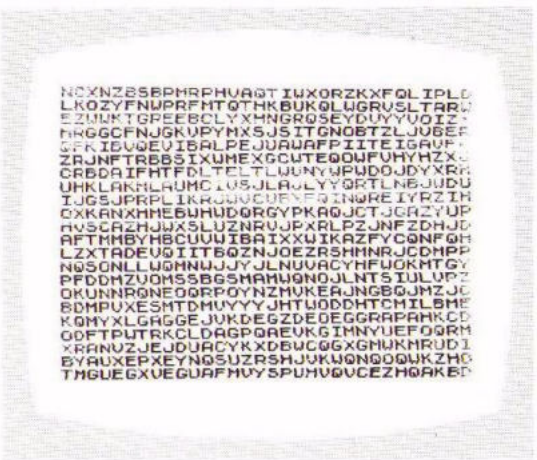
```
10 PRINT CHR$(32+INT(RND*224));
20 GOTO 10
```

Het scherm wordt geheel willekeurig met letters en symbolen gevuld.



Het zou leuk zijn als we als volgende stap willekeurige woorden zouden kunnen schrijven. In het printje, dat als gevolg van het bovenstaande programma ontstaat, zien we hier en daar al woorden zoals ''GOTO'', ''COPY'', enzovoorts. Dat zijn uiteraard de BASIC- sleutelwoorden, die we ook op het toetsenbord terugvinden en die door de computer als enkele symbolen worden geïnterpreteerd. Ze worden in de computer geschreven met één toetsdruk en worden in één geheugenlokatie bewaard. Het enige verschil met een gewone letter is dat de computer de symboolcode van deze sleutelwoorden vertaalt naar een aantal letters, als hij ze op het scherm moet schrijven.
Het zal duidelijk zijn dat het zeer moeilijk is andere dan deze vast ingeprogrammeerde sleutelwoorden door een computer te laten opwekken. Als u een computer willekeurige letters laat produceren (alle andere symbolen en sleutelwoorden worden uit het programma verwijderd) dan mag u blij zijn, als u af en toe een bestaand woord van drie of vier letters ziet verschijnen. Probeer het maar eens!

```
10 PRINT CHR$(INT(RND*26)+65);
20 GOTO 10
```



Het feit dat een computer niet uit zichzelf woorden kan produceren, beperkt het toepassingsgebied van zo'n machine bij een heleboel spelletjes. Als u bijvoorbeeld een getallen-raadspel wilt ontwerpen, kunt u de computer willekeurige getallen laten produceren en nadien trachten deze getallen te raden. Dit gaat niet op voor spelletjes, waarvan het doel is woorden te raden. De computer kan geen woorden maken. Er blijft dus niets anders over dan een heleboel woorden in het geheugen van het apparaat in te

voeren en de Spectrum op goed geluk woorden uit deze voorraad op het scherm te laten schrijven. Het raden van woorden uit een reeks die u net zélf in het geheugen van de machine hebt gezet, is natuurlijk niet leuk. Vandaar dat of iemand anders de woorden moet intikken, of dat er zoveel woorden in het geheugen worden geschreven, dat u ze onmogelijk allemaal kunt onthouden.

**Raad een woord**

Het volgende programma beschrijft een eenvoudig spel, waarbij het de bedoeling is een woord te raden. Dat spel heet in het Engels "hangman", (Poppetje aan de galg).
Wel moeten we een beroep op iemand anders doen, die zonder dat de speler het ziet een viertal woorden in de computer intikt.
Het programma is vrij lang:

```
10 LET w$="<"
20 FOR i=1 TO 4
30 INPUT a$
40 LET w$=w$+a$+"<"
50 NEXT i
60 CLS
70 PRINT "GALGJE"
80 LET r=INT(RND*LEN w$-1)+1)
90 IF w$(r)="<" THEN GOTO 120
100 LET r=r-1
110 GOTO 90
120 LET a$=""
130 w$=w$(1 TO r-1)+w$(r+1 TO )
140 IF w$(r)="<" THEN GOTO 170
150 LET a$=a$+w$(r)
160 GOTO 130
170 FOR i=1 TO LEN a$
180 PRINT "*";
190 NEXT i
200 LET h=0
210 PRINT AT 3,0;"Raad een letter"
220 INPUT b$
230 LET k=0
240 FOR i=1 TO LEN a$
250 IF b$(1)=a$(i) THEN LET k=i
260 NEXT i
270 IF k=0 THEN GOTO 210
280 LET a$(k)="*"
290 LET h=h+1
300 PRINT AT 1,k-1;b$(1)
310 IF h<>LEN a$ THEN GOTO 210
320 PRINT AT 4,0;"Geraden"
330 PAUSE 100
340 IF LEN w$>1 THEN GOTO 60
```

Het programma start met het intoetsen van vier woorden (regels 10 tot en met 60). Ieder woord wordt opgeslagen in een string, voorgesteld door "w$". Tussen ieder woord staat een "<"-teken, zodat de computer de woorden van elkaar kan onderscheiden.
Ter controle kan eventueel een extra regel "45 PRINT w$" worden opgenomen.
Nadat het scherm met regel 60 is gewist, start het eigenlijke spel. Eerst moet er door de machine een woord uit de lijst worden gekozen, hetgeen gebeurt aan de hand van de regels 80 tot en met 160. Met regel 80 wordt een willekeurig getal opgewekt, dat kleiner is dan het aantal letters in de lijst "w$". Dit getal wijst als het ware het geselecteerde woord aan. Nadien wordt dit uitgekozen woord in een nieuwe string "a$" opgenomen en verwijderd uit de oude string, zodat het niet per ongeluk nog eens een keer kan worden uitgekozen. Dat doen we door eerst de "aanwijzer" "r" naar de eerste "<"te verplaatsen en daarna alles van daaruit naar de volgende "<" in "w$" te verschuiven. Nadat de computer het te raden woord uit de lijst heeft getrokken, schrijft hij voor iedere letter uit dat woord een sterretje op het scherm (regels 170-190).
Dan wacht het programma, tot u een letter intoetst (regel 220).
Uw gok-letter wordt vergeleken met het geselecteerde woord door middel van de "FOR"-lus in de regels 240-260. Als de door u gegokte letter in het woord voorkomt, wordt de plaats van deze letter in het woord opgeslagen in de variabele "k". Door de instructies in de regels 270 tot en met 310 wordt de letter op de korrekte plaats in het woord op het scherm geschreven en het sterretje op die plaats gewist. Een en ander heeft ook tot gevolg, dat de gegokte letter niet meer in aanmerking komt als goed antwoord in volgende pogingen.
Er wordt nu opnieuw de vraag "Raad een letter" op het scherm geschreven, u gokt een tweede letter.
Als het aantal geraden letters overeenkomt met de lengte van het te raden woord, besluit de computer dat u het woord helemaal hebt geraden en dat een felicitatie op zijn plaats is. Regel 320 schrijft "goed geraden" op het scherm!
Na een pauze (regel 330) zal de computer een nieuw woord uitkiezen (als er nog een voorradig is) en een nieuw spel kan beginnen.
In dit programma worden een aantal zeer interessante programmeer-kunstjes toegepast. Neem dus even de moeite om het goed te bestuderen en probeer iedere regel te begrijpen.
Nadien kunt u dit programma ingewikkelder maken, door bijvoorbeeld niet 4 maar 100 woorden in te tikken (U kunt het spel dan zelf spelen, want zo veel woorden kunt u toch niet

onthouden!). Daarnaast kunt u ook proberen het spel uit te breiden met enige grafische instructies en een overzichtelijk systeem voor het op het scherm schrijven van het aantal beurten waarin het woord is geraden.

**Codes en cijferschriften**

Computers kunnen goed overweg met zowel cijfers als letters. Het ligt voor de hand dat een computer een uitstekend werktuig is voor diegenen die geïnteresseerd zijn in het coderen en decoderen van codes en cijferschriften.

Tijdens de tweede wereldoorlog is heel wat geheime informatie uitgelekt, doordat computers in staat waren gedecodeerde berichten te ontcijferen.

Echte codes ontcijferen is te veel gevraagd van onze kleine Spectrum. Door gebruik te maken van de "RND"- en "RAND"-instructies kunnen we de computer echter wel omvormen in een simpele codeer- en decodeermachine.

Als voorbeeld gaan we onderstaand programma bespreken.

```
10 PRINT "CODEER"
20 PRINT "Wat is uw codesleutel?"
30 INPUT k
40 RAND k
50 PRINT "Ontcijfer of decodeer (0/1)"
60 INPUT d
70 IF d=0 THEN LET d=-1
80 PRINT "Uw boodschap"
90 INPUT a$
100 FOR i=1 TO LEN a$
110 LET a=CODE a$(i)-64
120 IF a$(i)="" THEN LET a=0
130 LET a=a+d*INT(RND*59)
140 LET a=ABS(a-INT(a/59)*59)
150 IF a=0 THEN LET a=-32
160 PRINT CHR$(a+64);
170 NEXT i
```

Test het programma uit door het de volgende code-boodschap te laten ontcijferen:

WqMhBhV1U£x ZeILi

Op de vraag "wat is uw codesleutel" moet 1983 worden geantwoord. Reageer daarna met een "0" op de codeer/decodeer-vraag en toets de te decoderen boodschap in. De computer zal de gedecodeerde tekst op het scherm schrijven.

Dit programma maakt gebruik van een "RAND"-eigenschap, namelijk dat de volgorde van de willekeurig berekende getallen vast ligt (zie hoofdstuk 3). U hoeft alleen maar dezelfde "RAND"-waarde in te toetsen! Vergeet echter niet dat "RAND 0" het startpunt van de reeks willekeurige getallen bepaalt aan de hand van het aantal geschreven TV-beelden. Deze code kunnen we dus niet gebruiken bij dit programma, want de reeks willekeurige getallen zou dan niet te herhalen zijn.

Met regel 30 vraagt de computer de door u gekozen code op. Deze code wordt in regel 40 gebruikt voor het starten van de reeks van willekeurige getallen. Dezelfde code wordt ook gebruikt voor het decoderen van de boodschap. Als u de code niet kent, is het onmogelijk het bericht te ontcijferen!

Het bericht dat via regel 90 wordt ingetoetst, wordt opgesplitst in individuele letters en iedere letter wordt door de "CODE"-instructie omgezet in een getal. Van dit getal wordt steeds een waarde van 64 afgetrokken, zodat er geen grafische symbolen in de gecodeerde boodschap verschijnen. Weliswaar zou het gebruik van grafische symbolen niets afdoen aan de werking van het systeem, maar dit soort tekens is vrij onpraktisch, als we de gecodeerde mededeling willen uittikken met een normale schrijfmachine!

Regel 120 geeft een vaste code "0" aan een spatie. De overige letters worden gecodeerd door eerst een willekeurig getal tussen 0 en 59 bij het CODE-getal op te tellen, het resultaat te delen door 59 en de rest van deze bewerking verder uit te werken. Deze rest ligt namelijk in hetzelfde gebied als de symboolcode waarmee we gestart zijn, dus tussen 0 en 58.

Door de "CHR$(a+64)"-instructie zullen de letters, die we op deze manier hebben berekend, op het scherm worden geschreven. Daarbij houden we rekening met de code voor de spatie, door het invoeren van regel 150.

Bij het opnieuw decoderen van de boodschap, wordt het bepaalde willekeurige getal tussen 0 en 59 weer van de berekende code afgetrokken, waardoor de originele symboolcode ontstaat.

Door middel van de variabele "d" kunnen we de functie van dit programma definiëren: "d" = —1 voor decoderen, "d" = 1 voor coderen.

Dit codeerprogramma is vrij kort en tamelijk eenvoudig, maar levert toch goede resultaten, die moeilijk te ontcijferen zijn zonder kennis van de code. Zonder deze sleutelcode kan men echt niet uit de voeten met een door de Spectrum gecodeerde boodschap, omdat een en dezelfde letter verschillende symbolen kan voorstellen, afhankelijk van de plaats in de tekst.

**Een eenvoudige vorm van Master-Mind**

We kunnen met de Spectrum een spelletje ontwerpen, waarbij het er op aan komt zowel de juiste cijfers uit een getal te raden, als de juiste volgorde van de cijfers in het getal. Een soort vereenvoudigde uitvoering van Master-Mind dus. We moeten dan wel twee problemen kunnen oplossen. In de eerste plaats moeten we in staat zijn individuele cijfers uit het geraden getal te vergelijken met individuele cijfers uit het te raden getal. In de tweede plaats moeten we in staat zijn willekeurige getallen met een gegeven aantal cijfers op te wekken.

De afzonderlijke cijfers in de getallen kunnen we door gebruik te maken van strings op de gewenste manier manipuleren.

Het onderstaande programma werkt als volgt. De Spectrum genereert een willekeurig getal van vier cijfers, met dien verstande dat een cijfer slechts éénmaal wordt gebruikt. Nadien raadt u een getal van vier cijfers en de computer vertelt u:

a - hoeveel cijfers u juist hebt geraden;

b - hoeveel cijfers op de juiste plaats staan.

In dit programma noemen we een cijfer, dat op de juiste plaats staat een ''plaats'' en een cijfer dat wel in het te raden getal voorkomt, maar op een verkeerde plaats is geraden een ''geraden cijfer''.

Een voorbeeldje. Stel, dat de computer het getal 1234 uitkiest. U raadt 2035. U heeft dan een ''plaats'' geraden, de 3, en een ''geraden cijfer'', de 2.

Om problemen met het definiëren van het aantal ''geraden cijfers'' te vermijden, werken we met getallen waarin ieder cijfer slechts één keer wordt gebruikt.

```
10 LET a$="":LET g=0
20 RAND 0
30 FOR i=1 TO 4
40 LET b$=STR$ INT(RND*10)
50 FOR j=1 TO LEN a$
60 IF a$(j)=b$ THEN GOTO 40
70 NEXT j
80 LET a$=a$+b$
90 NEXT i
100 INPUT b$
110 IF LEN b$<>4 THEN BEEP.5.0:GOTO 100
120 LET p=0:LET h=0
130 LET g=g+1
140 FOR i=1 TO 4
150 IF a$(i)=b$(i) THEN LET p=p+1
160 FOR j=1 TO 4
170 IF a$(j)=b$(i) THEN LET h=h+1
180 NEXT j
190 NEXT i
200 LET h=h-p
210 PRINT "Raden ";b$
220 PRINT "Juiste plaats = ";p,"goed geraden = ";h
230 IF a$<>b$ THEN GOTO 100
240 PRINT "Geraden in ";g;" beurten"
```

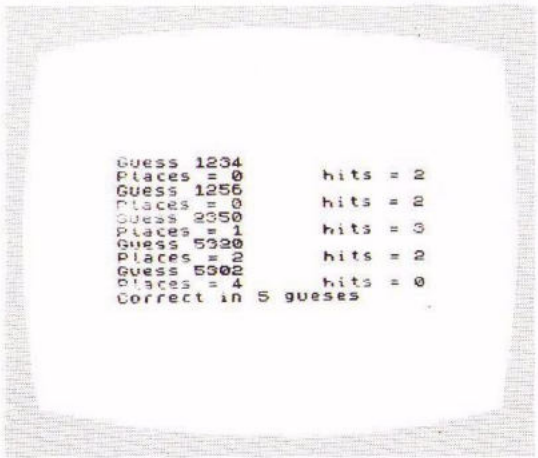
Het willekeurige getal van vier cijfers wordt gegenereerd aan de hand van de instructies in de regels 30 tot en met 90. Een willekeurig cijfer wordt als een string geproduceerd met instructie 40 en de gegenereerde cijfers worden met elkaar vergeleken in de regels 50 tot 70.

Wordt er een cijfer berekend dat al aanwezig is, dan wordt dit geweigerd, de computer springt terug naar regel 40 en waagt een nieuwe poging. Is het cijfer nog niet aanwezig, dan wordt het bij het getal in regel 80 gevoegd.

Na regel 100 wacht het programma tot u een getal van vier cijfers intoetst. Als u te veel of te weinig cijfers intoetst, wordt uw getal geweigerd. De computer wekt een toontje op, springt terug naar regel 100 en wacht op een korrekte invoer.

De programma-regels 140 tot en met 190 vergelijken uw getal met het door de computer gemaakte. Het aantal ''plaatsen'' wordt bepaald door beide getallen cijfer voor cijfer te vergelijken in regel 150 en voor iedere ''plaats'' de waarde van ''p'' met een eenheid te verhogen. Het controleren van het aantal ''geraden cijfers'' is niet zo eenvoudig en vergt een extra ''FOR''-lus (regels 160 tot 180). Ieder cijfer van uw getal wordt vergeleken met ieder cijfer in het computergetal en als er een gelijkheid wordt aangetroffen, zal de variabele ''h'' met een eenheid worden verhoogd. Natuurlijk tellen we met deze methode ook het aantal ''plaatsen'' op bij het aantal ''geraden cijfers''. Vandaar dat we in regel 200 de ''p''-waarde aftrekken van de net berekende ''h''-waarde. Deze nieuwe ''h'' geeft het aantal ''geraden cijfers''. De twee volgende regels van het programma verzorgen het op het scherm schrijven van de nodige mededelingen. Regel 230 bepaalt of u er in bent geslaagd het computergetal volledig en in de juiste volgorde te raden. Is dat het geval, dan schrijft regel 240 het aantal beurten op het scherm.

In het hier naaststaande voorbeeld had de computer het getal 8340 uitgekozen en kostte het de speler 5 beurten om dit getal te raden.




Zoals reeds in de inleiding van deze paragraaf gesteld, geeft dit programma een zeer eenvoudige uitvoering van het "Master-Mind"-spel. Niets belet u echter dit programma te perfectioneren door bijvoorbeeld met kleur en geluid te gaan werken.
Maar zelfs zonder deze uitbreidingen is dit spel vrij verslavend!

# 10. GRAFISCHE MOGELIJKHEDEN VOOR GEVORDERDEN

Vaak kunnen we eenvoudigere of effectievere programma's schrijven als we gebruik maken van technieken met zowel lage als hoge resolutie.
Daar is echter één groot nadeel aan verbonden: de twee verschillende coördinatensystemen!
In dit hoofdstuk gaan we ons bezig houden met het schrijven van programma's, waarin beide methoden door elkaar worden gebruikt.

**Een duikboot-spel**

Met dit programma gaan we twee schepen op het scherm tot leven brengen, namelijk een slagschip en een onderzeeër. Het slagschip kan dieptebommen lanceren. Het is uiteraard de bedoeling dat de duikboot wordt geraakt.
De vormen van beide schepen kunnen we zeer eenvoudig met de bekende technieken met lage resolutie opbouwen. Het lanceren van de dieptebommen, daarentegen, kunnen we het best met technieken met hoge resolutie nabootsen.
Aan de hand van alles wat we in hoofdstuk 6 hebben uitgelegd, moet het u lukken dit programma te schrijven. Het ligt voor de hand dat we het varen van de schepen nabootsen door eerst op een bepaalde positie het symbool te tekenen, het nadien te wissen, de coördinaten iets te veranderen en het op deze nieuwe positie opnieuw te tekenen. Omdat beide schepen slechts horizontaal in een rechte lijn bewegen, kunnen we een trucje toepassen, waardoor dit deel van het programma wordt vereenvoudigd.
Als simpel voorbeeld van dit trucje:

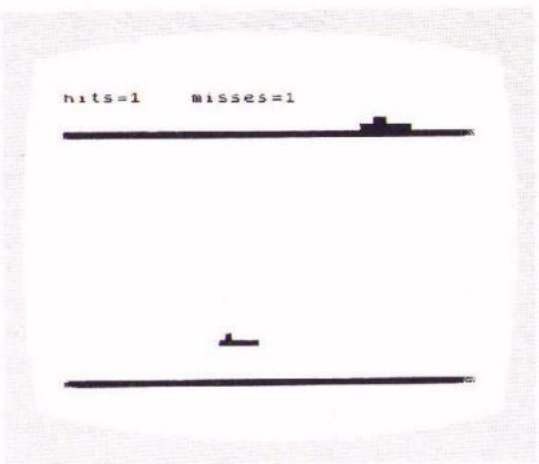
```
10 FOR x=0 TO 30
20 PRINT at 3,x;"[8]*";
30 NEXT x
```

Er zal een sterretje van links naar rechts over het scherm bewegen. Het tekenen, wissen en opnieuw tekenen van het objekt wordt hierbij in één regel gerealiseerd, door het "spatie"-symbool in het "PRINT"-bevel op te nemen. Deze techniek kunnen we meestal gebruiken, als we er maar voor zorgen dat we voldoende spaties rond het objekt opnemen. Het volledige objekt moet namelijk bij de volgende "PRINT" worden vervangen door spaties.
We zeggen "meestal", omdat dit systeem bij niet-rechtlijnige bewegingen erg moeilijk in praktijk is te brengen.
Deze methode hebben we overigens al eerder toegepast, namelijk bij het opbouwen van het slaghout bij het "squash"-spel. De grootste problemen bij dit programma ontstaan bij het bepalen of de duikboot wel of niet is geraakt en bij het nabootsen van een soort "explosie", waarmee de getroffen duikboot van het scherm moet verdwijnen.
Zoals reeds gezegd in de inleiding, is een extra complicatie het feit dat de boten worden opgebouwd volgens technieken met lage resolutie en de dieptebom volgens technieken met hoge resolutie, elk met hun eigen coördinatenstelsel.
Het volledige programma, met grafische symbolen voorgesteld door rechte haakjes rond het in te drukken symbool, is als volgt:

```
10 LET mc=0
20 LET hc=0
30 LET f=0
40 LET h=0
50 LET xs=0
60 GOSUB 600
70 LET x=0
80 LET ys=INT(RND*5)+15
90 LET y=2
100 GOSUB 200
110 PRINT AT 0,0;"Treffers=";hc;TAB(10);
    "Missers=";mc
120 IF f=0 THEN LET xd=8*x:LET yd=140
130 REM f is de vuurindicatie
140 IF INKEY$="f" AND f=0 THEN LET f=1
150 IF f=1 THEN GOSUB 300
160 GOSUB 400
170 IF h=0 THEN GOTO 100
180 GOSUB 500
190 GOTO 30

200 PRINT AT y,x;"[8][^3][^8][^3][^3]";
210 LET x=x+1
220 IF x>25 THEN LET x=0
230 IF x=0 THEN PRINT AT y,26;"[8][8][8][8]";
240 RETURN

300 PLOT INVERSE 1,xd,yd
310 LET yd=yd-4
320 PLOT xd,yd
330 REM raak?
340 IF yd<16 THEN LET f=0:LET mc=mc+1
350 IF f=0 THEN PLOT INVERSE 1,xd,yd
360 IF ABS(xd-8*xs-24)>24 THEN  RETURN
370 IF ABS(yd-175+8*ys)>8 THEN  RETURN
380 LET h=1
390  RETURN
```

```
400 PRINT AT ys,xs;"[8][^2][^3][^3]";
410 LET xs=xs+RND
420 IF xs>28 THEN LET xs=0
430 IF xs=0 THEN PRINT AT ys,28;"[8][8][8][8]";
440 RETURN

500 LET hc=hc+1
510 FOR i=1 TO 20
520 PRINT AT ys,xs;"[8][^4][4][5]";
530 PRINT AT ys,xs;"[8][8][8][8]"
540 NEXT i
550 RETURN

600 CLS
610 FOR i=0 TO 31
620 PRINT AT 3,i;"[^3]";
630 PRINT AT 21,i;"[3]";
640 NEXT i
650 RETURN
```

Eerst worden een aantal variabelen op nul gezet (geïnitialiseerd, heet dat in vaktermen) en door middel van subroutine 600 de bodem en het oppervlak van de zee op het scherm getekend.
Nadien wordt de vaardiepte van de onderzeeër door middel van een ''RND''-functie op een willekeurige waarde vastgelegd (regel 80), het oppervlakte-schip op startpositie ''x, y'' getekend (regel 200) en de duikboot op startpositie ''xs, ys'' ingevuld (subroutine 400).




De duikboot beweegt in dezelfde richting als het vaartuig met een willekeurige snelheid, bepaald door regel 410. Door het gebruik van de ''INKEY$''-functie kunnen we controleren of de toets ''F'' wordt ingedrukt (regel 140). Is dat het geval, dan wordt de dieptebom gelanceerd. De variabele grootheid ''f'' wordt 1 en de positie van de dieptebom wordt opgeslagen in de variabelen ''xd'' en ''yd''. Door middel van subroutine 300 blijft de dieptebom vallen en blijft de computer op de hoogte van de juiste coördinaten van dit helse tuig.
Hier krijgen we te maken met de twee coördinaten-stelsels: de dieptebom verschijnt op het scherm als gevolg van een ''PLOT''-instructie, de schepen als resultaat van ''PRINT AT''-bevelen.
''PLOT'' verdeelt het scherm in 256 bij 176 punten, ''PRINT AT'' werkt met 32 bij 22 symboolplaatsen. Bovendien worden ''xd'' en ''yd'' gemeten vanaf de onderzijde van het scherm en ''y'' vanaf de bovenkant!
Vandaar dat zowel het lanceren van de bom vanaf het schip als het detecteren van een treffer niet zo eenvoudig is als het lijkt!
Met de regels 120 en 130 worden de startcoördinaten van de bom (''xd'' en ''yd'') gelijk gemaakt aan de coördinaten van het schip. Zo zijn we er zeker van dat de bom steeds vanaf het slagschip wordt gelanceerd.
Met de regels 360 en 370 vergelijken we de coördinaten van de bom met die van de onderzeeër. Het zal duidelijk zijn, dat we de x-coördinaten van de duikboot met acht moeten vermenigvuldigen. Een symboolplaats is immers opgebouwd uit 8 punten en de x-coördinaten lopen beide van links naar rechts.
Het vergelijken van de y-coördinaten is iets moeilijker: we moeten de waarde ''8*ys'' aftrekken van het getal 175. De overige programmastappen zijn recht-toe-recht-aan. Noteer de manier waarop we de onderzeeër ''vernietigen'' met de regels 510 tot en met 540. Dit kan in andere spelletjes nog van pas komen!

**De plattegrond van het scherm**

Alles wat op het TV-scherm verschijnt, wordt opgeslagen in het geheugen van de Spectrum. Weten waar en hoe dat gebeurt is erg nuttig, want we kunnen er nieuwe systemen uit afleiden om te werken met grafieken.
Zoals reeds in hoofdstuk 2 is geschetst, wordt de informatie waarmee het beeld wordt opgebouwd in twee delen gesplitst. Een deel bevat de informatie over het inkt- of papierkarakter van een punt en dat deel noemen we dus het inkt-papiergeheugen, een ander deel bevat de informatie over de attributen van een punt en dat deel noemen we het attributengeheugen.
Het inkt-papiergeheugen bevat de informatie, waaruit de machine kan afleiden welke van de

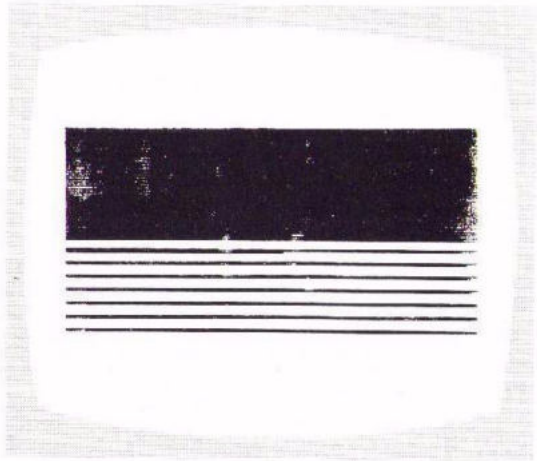
256 bij 176 schermpunten papier- of inktpunten zijn. Het attributengeheugen bewaart de "INK", "PAPER"-, "BRIGHTNESS"- en "FLASH"-gegevens van ieder punt.

Laten we dieper ingaan op beide geheugens. Het inkt-papierdeel van het geheugen gaat van geheugenadres 16384 tot 22327. Iedere geheugencel kan de informatie van 8 punten bewaren. Het verband tussen een plaats in het geheugen en een plaats op het scherm is tamelijk gecompliceerd. Als we beginnen aan de bovenzijde van het scherm, worden de 8 meest linkse punten opgeslagen in de eerste geheugencel. De volgende 8 horen thuis in de tweede cel en zo verder voor de 256 punten van de eerste rij. De eerste 32 geheugenplaatsen bewaren dus de informatie van alle punten op de bovenste rij van het scherm. Omdat een symboolplaats is opgebouwd uit 8 punten, kunnen we stellen dat deze geheugencellen de informatie bevatten van de bovenste rij punten van de bovenste regel.
Logischerwijze zou men verwachten dat de volgende 32 geheugencellen de gegevens huisvesten van de punten van de tweede rij. Jammer genoeg is dat niet het geval. In deze cellen bewaart men de gegevens van de bovenste rij punten van de tweede regel, of met andere woorden de gegevens van de negende rij punten.
Nadien gaat men verder met de eerste rij punten van de derde regel. Deze struktuur wordt herhaald tot en met de achtste regel.
Dan begint men van voren af aan, maar nu met de tweede rij punten van de eerste regel. Als alle rijen punten van de bovenste 8 regels in het geheugen zijn ondergebracht, start men volgens hetzelfde systeem met de punten van de negende tot en met 16-de regel. De 24 regels worden dus onderverdeeld in drie blokken van ieder 8 rijen.
Deze geheugenindeling is vrij moeilijk te begrijpen, maar door middel van een eenvoudig programmaatje kunnen we dit proces zichtbaar maken op het scherm.

```
10 FOR i=16384 TO 22527
20 POKE i,255
30 NEXT i
```

Met dit programma maken we de inhoud van alle geheugencellen gelijk aan "1" (255= BIN 11111111) en wel vanaf geheugenadres 16384 tot adres 22527.

Uit de volgorde waarmee het scherm wordt volgeschreven met inktpunten kunnen we de plaats van ieder punt in het geheugen afleiden.



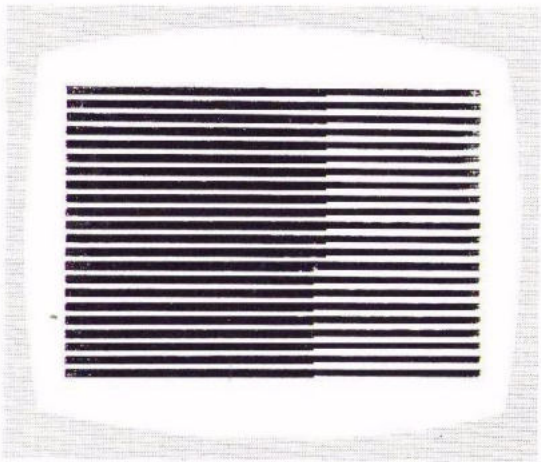
Uit dit programma kunnen we nog een belangrijk gegeven afleiden, namelijk dat een binaire "1" overeenkomt met een inktpunt en een binaire "0" met een papierpunt.
Dat is allemaal leuk en aardig, maar veel hebben we niet aan deze wetenschap, als we niet in staat zijn door middel van een simpele formule de plaats van een punt op het scherm rechtstreeks te koppelen aan een geheugenadres!
We moeten dus een formule opstellen, waarin we de lijn-, kolom- en symboolrij-volgnummers van één punt kunnen invullen en waaruit we het adres van de geheugenlokatie van dat punt kunnen afleiden.
De volgende ingewikkelde formule voldoet aan die eis:

```
geheugenlocatie =16384+2048*INT(L)
                +32*(L-8*INT(L/8))+256*R+C
```

Hierin staat "L" voor lijn-nummer, "C" voor kolom-nummer en "R" voor het symboolrij-nummer, alle drie startend vanaf 0. Als u bijvoorbeeld wilt weten welk puntenpatroon is opgeslagen in de derde rij van het tweede symbool in de derde lijn, dan moet u "R" gelijk stellen aan 2, "C" gelijk aan 1 en "L" gelijk aan 2.
U kunt deze formule aan het werk zien door het volgende programma in te toetsen.

```
10 DEF FNs(L,c,r)=16384+2048*INT(L/8)
   +32*(L-8*INT(L/8))+256*r+c
20 CLS
30 FOR n=0 TO 7
40 FOR i=0 TO 31
50 FOR j=0 TO 23
60 POKE FNs(i,j,n),255
70 NEXT j
80 NEXT i
90 NEXT n
```

Het scherm wordt van boven naar onder en van links naar rechts volgeschreven met korte horizontale lijnen.



Hoewel het dus zonder meer mogelijk is, zal het ingewikkelde systeem waarmee de schermpunten in het geheugen zijn opgeslagen u er waarschijnlijk van weerhouden om door middel van "PEEK" en "POKE" rechtstreeks in te grijpen in dit gedeelte van het geheugen.
Laten we nu eens gaan kijken naar het attributengeheugen. Gelukkig is dit veel logischer ingedeeld. Vanaf adres 22528 tot adres 23295 staat er één geheugenlokatie ter beschikking voor iedere symboolplaats op het scherm.
Ook het verband tussen een plaats op het scherm en een plaats in het geheugen is vrij logisch. De eerste symboolplaats op de eerste schermlijn komt overeen met de eerste geheugencel, de tweede symboolplaats met de tweede geheugencel en zo verder tot aan het einde van de lijn.
Nadien starten we op dezelfde manier met het onderbrengen van de symbolen van de tweede lijn, dus het eerste symbool van de tweede lijn komt overeen met de 33-ste geheugencel. Dat gaat zo verder tot aan de onderkant van het scherm. We kunnen dus een simpele formule opstellen, die het verband aangeeft tussen het geheugenadres en de symboolplaats:

```
22528+32*L+C
```

Hierbij staat "L" voor het lijnnummer en "C" voor het kolomnummer, beide variabelen starten vanaf de waarde nul. Iedere geheugencel bewaart de momentele gegevens van de attributen van elke symboolplaats. Hoe dat gebeurt, zullen we later bespreken. U kunt nu alvast het volgende programma uitproberen.

```
10 FOR i=1 TO 32*21
20 PRINT CHR$(INT(RND*26+65));
30 NEXT i
40 FOR x=22528 TO 23295
50 POKE x,INT(RND*256)
60 NEXT x
70 GOTO 40
```

Het eerste gedeelte van dit programma, namelijk de regels 10 tot en met 30, vult het scherm met willekeurig gekozen symbolen. Het tweede deel verandert de gegevens van de attributen eveneens op een willekeurige manier en wel van links boven naar rechts onder.
Over dit programma kunnen we twee opmerkingen maken. In de eerste plaats blijkt uit de beeldopbouw en de manier waarop de symboolplaatsen van eigenschappen veranderen, het verband tussen geheugenplaats en symboollokatie. In de tweede plaats blijkt dat alleen dat wat op het scherm is geschreven, wordt aangepast aan de nieuwe attributen. Er wordt niets bijgeschreven en er valt niets weg, dat wat er wél staat gaat knipperen of verandert van kleur. Dit kan erg nuttig zijn, als we bijvoorbeeld de kleur van een grafiek willen veranderen, dan hoeven we niet alles opnieuw op het scherm te schrijven.
Wat we echter nog steeds niet weten is hoe de attribuutgegevens van een symboolplaats in een geheugencel zijn gerangschikt.
De onderstaande formule berekent het getal dat hoort bij een bepaalde set attributen:

```
128*FLASH+64*BRIGHT+8*PAPER+INK
```

"FLASH" en "BRIGHT" zijn 0 of 1 (bij de laatste waarde gaat het symbool knipperen of helder oplichten), "PAPER" en "INK" zijn gelijk aan de in hoofdstuk 2 besproken cijfer-

kleuren-code. Als u dus ergens een knipperend rood symbool op een zwarte achtergrond met normale helderheid op het scherm wil schrijven, wordt de inhoud van de geheugencel:

```
128*1+64*0+8*0+2
```

wat dus gelijk is aan 130.
Om het leven van de programmeur wat gemakkelijker te maken, beschikt de Spectrum over een instructie, waarmee we zonder "PEEK"-omwegen de attributen van een geheugenplaats op het scherm kunnen schrijven:

```
ATTR(lijn,kolom)
```

Natuurlijk moeten we dan wel in staat zijn, de inhoud te vertalen in de juiste "INK"- en "PAPER"-kleuren en de exacte waarde van "FLASH" en "BRIGHT".
Hiervoor kunnen we de vier onderstaande formules gebruiken.

```
flash:  INT(ATTR(line,col)/128))
bright: INT((ATTR(line,col)-INT(ATTR(line,col)/
        128)*128)/64)
paper:  INT((ATTR(line,col)-INT(ATTR(line,col)/
        64*64)/8)
ink:    ATTR(line,col)-INT(ATTR(lin,col)/8)*8
```

**De instructies "SCREEN$" en "POINT"**

Nu we weten hoe alles wat op het scherm geschreven staat in het geheugen wordt bewaard, zal ons wel een ding duidelijk zijn geworden. Terugvinden van bijvoorbeeld de kleur van een punt, door het bekijken van de inhoud van het geheugen gaat tamelijk gemakkelijk. Veel moeilijker is het om aan de hand van de geheugeninhoud een idee te krijgen van wat er op het scherm geschreven staat. Zelfs door gebruik te maken van de beschreven formule kunnen we door het geheugen te "PEEK"-en toch niet meer te weten komen dan hoe de verdeling van inkt- en papierpunten is in de 64 punten, die een symboolplaats vormen.
Aan de hand van deze informatie kunnen we natuurlijk wel bepalen hoe het symbool er uit ziet, maar dat alles is verschrikkelijk omslachtig en nodigt niet uit om er vaak gebruik van te maken.
Gelukkig komt de Spectrum ons te hulp met zijn (SCREEN$ "lijn", "kolom")-functie. Deze functie zal het patroon van 64 punten van de geselecteerde symboolplaats vergelijken met alle puntenpatronen die in de symbool-generator tabel zijn opgeslagen (zie hoofdstuk 7).
Als het onbekende patroon overeenkomt met het patroon van een in de tabel opgenomen symbool, dan zal de "SCREEN$"-functie dat symbool als een reeks op het scherm zetten.
Als het onbekende patroon niet in de tabel voorkomt, dan zal de computer een 0-reeks als antwoord geven. Zonder twijfel een handig systeem en bovendien werkt dit grapje altijd, onafhankelijk van hoe het puntenpatroon in het geheugen is terechtgekomen. Zo kunt u door een "PLOT"-instructie het puntenpatroon van de letter "A" op de juiste geheugenplaatsen invoeren. Zolang ze daar blijven, kunnen we door middel van de "SCREEN$"-functie altijd de letter A terugroepen.
Een tweede spitsvondigheid van deze functie is dat ze geen verschil ziet tussen een symbool en het geïnverteerde symbool! Dat betekent bijvoorbeeld dat zowel een spatie [CHR$ (32)], als het volle vierkant [^8], als het lege grafische symbool [8] bij onderzoek van hun puntenpatroon met de "SCREEN$"-functie een spatie terug gaan! Een voorbeeld van het gebruik van deze functie vindt u bij de bespreking van het doolhofspel. Er bestaat ook een instructie, waarmee we zonder omwegen kunnen vaststellen of een schermpunt als inkt- of papierpunt is geprogrammeerd.
De instructie:

```
PUNT(x,y)
```

antwoordt met 0 als het punt met coördinaten "x, y" een papierpunt is en met 1 als het betreffende punt als inktpunt is geprogrammeerd. Hoewel deze instructie dus zeer eenvoudig is toe te passen, zullen we er in de praktijk toch niet zo vaak naar teruggrijpen. Het is namelijk meestal niet interessant om te weten wat de functie van een enkel punt is.

**Een doolhofspel**

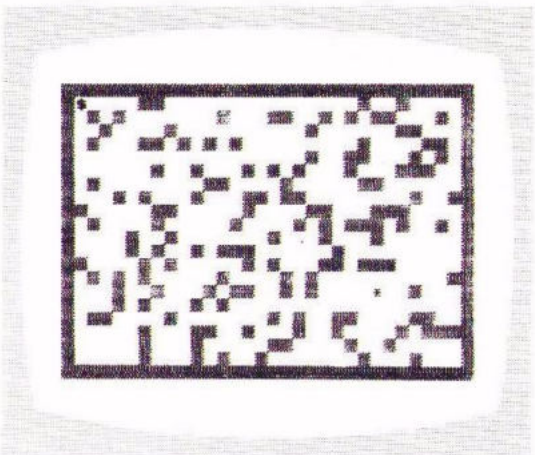
Dit doolhofspel is de Spectrum-versie van een zeer populair spelletje, dat we bij zowat alle microcomputers tegenkomen.
Bij dit spel komt het er op aan een sterretje van de rechter onderhoek van het scherm naar

boven te sturen, door een steeds wisselend patroon van hindernissen. De kunst is natuurlijk het sterretje zo snel mogelijk de openingen tussen hindernissen door te sturen.
Alvorens naar het onderstaande programma te kijken, moet u eerst zelf eens proberen zo'n programma te schrijven. Het grootste probleem is blijkbaar dat deel te schrijven, waardoor de computer kan bepalen wanneer uw weg door een hindernis wordt geblokkeerd. Daarnaast moet u zich afvragen of het echt noodzakelijk is alle coördinaten van alle hindernissen in het programma op te nemen. Als u er niet uitkomt, of uw programma wil vergelijken met het onze, bestudeer dan de volgende pagina's.

```
10 INK 2:PAPER 6:BORDER 1
20 INPUT "Moeilijkheidsgraad 1-9 ?";d
30 IF d<1 OR d>9 THEN GOTO 20
40 LET d1=d/10
50 CLS
60 GOSUB 100
70 GOSUB 300
80 GOSUB 500

100 POKE USR "a"+0,BIN 01010101

110 POKE USR "a"+1,BIN 10101010
120 POKE USR "a"+2,BIN 01010101
130 POKE USR "a"+3,BIN 10101010
140 POKE USR "a"+4,BIN 01010101
150 POKE USR "a"+5,BIN 10101010
160 POKE USR "a"+6,BIN 01010101
170 POKE USR "a"+7,BIN 10101010
180 RETURN

200 LET c$=SCREEN$(y,x)
210 RETURN

300 REM doolhofstructuur
310 GOSUB 1000
320 FOR i=1 TO 20*d+40
330 LET x=RND*29+1
340 LET y=RND*19+1
350 PRINT AT y,x,;"[a]";
360 NEXT i

370 PRINT AT 1,1;"$";
380 PRINT AT 20,30;"*";
400 RETURN

500 LET xc=30
510 LET yc=20
520 LET m=0
530 LET x=xc
540 LET y=yc
550 LET a$=INKEY$
560 GOSUB 900
570 IF a$="" THEN GOTO 550
580 IF a$="5" THEN LET x=xc-1
590 IF a$="8" THEN LET x=xc+1
600 IF a$="6" THEN LET y=yc+1
610 IF a$="7" THEN LET y=yc-1
620 GOSUB 200
630 IF c$="$" THEN GOTO 800
640 IF c$<>" " THEN BEEP .1,10:GOTO 530
650 PRINT AT yc,xc;" ";
660 LET xc=x
670 LET yc=y
680 PRINT AT yc,xc;"*";
690 LET m=m+1
700 GOTO 530
```

```
800 CLS
810 PRINT AT 0,2;"Aantal bewegingen ';m
820 PRINT "Nog een spel j/n"
830 INPUT a$
840 IF a$(1)="j" THEN RUN
850 IF a$(1)<>"n" THEN GOTO 820
860 STOP

900 IF RND>d1 THEN RETURN
910 LET c$=" "

920 IF RND>.5 THEN LET c$="[a]"
930 PRINT AT RND*19+1,RND*29+1;c$
940 PRINT at 1,1;"$";
950 RETURN

1000 FOT i=0 TO 31
1010 PRINT AT 0,i;INK 0;"[a]";AT 21,i;"[a]";
1020 NEXT i
1030 FOR i=0 TO 21
1040 PRINT AT i,0;INK 0;"[a]";AT i,31;"[a]";
1050 NEXT i
1060 RETURN
```

Het programma start met het opvragen van de gewenste moeilijkheidsgraad ''d''. Deze waarde bepaalt het aantal vierkantjes, waaruit de doolhof wordt samengesteld en dus ook het aantal hindernissen op uw weg.
Deze informatie wordt in subroutine 300 gebruikt om de doolhof op het scherm te tekenen.



Dit deel van het programma maakt gebruik van reeds zeer bekende technieken. Eerst wordt er een beroep gedaan op subroutine 1000, waarmee de rand rond de doolhof wordt getekend. Deze rand is samengesteld uit een U-D symbool, opgebouwd in subroutine 100.
Nadien wordt een ''PRINT AT''-instructie gebruikt, om het vierkantje (grafisch symbool [A]) op willekeurig bepaalde punten op het scherm te tekenen.

Tot slot tekent deze subroutine een "$"-teken in de linkerbovenhoek, waarmee we willen aangeven dat het de bedoeling is dat we het sterretje naar dit punt loodsen, dat nu in de positie "20, 30", dus rechtsonder staat. Het spel start met het oproepen van subroutine 500.
Eerst geven we aan een aantal variabelen een bepaalde aanvangswaarde (het zogenaamde initialiseren), daarna gaat de machine door middel van regel 550 bekijken of de "pijltjes"-toetsen al of niet worden ingedrukt.
Afhankelijk van welke toets men indrukt, zullen de oude coördinaten van het sterretje, "xc" en "yc", worden aangepast en het symbool wordt op de nieuwe positie "x, y" op het scherm geschreven.
Wél moeten we eerst bepalen of deze nieuwe positie wel kan. Als er op deze plaats niets op het scherm staat, is alles in orde. Als er echter wel iets op deze plaats staat, dan moet de nieuwe positie worden afgewezen.
Hoe doen we dat?
Vrij eenvoudig!
Met regel 620 roepen we de hulp in, van subroutine 200. Door middel van een "SCREEN$"-instructie gaat deze bekijken wat er op positie "x, y" in het geheugen zit en het resultaat opslaan in de variabele c$.
Zit er op die plaats een "$"-symbool, dan weten we dat we het sterretje naar de eindpositie hebben gestuurd en wordt het spel beëindigd door regel 800.
Staat er op lokatie "x, y" iets anders dan een spatiesymbool (wat door regel 640 wordt onderzocht), dan wordt er een kort toontje opgewekt met een "BEEP" en de "GOTO" zorgt ervoor, dat de coördinaten "x, y" weer gelijk worden aan "xc, yc".
Als er op positie "x, y" wél een spatiesymbool staat, dan wordt de oude sterretjes-positie gewist (regel 650) en de nieuwe geschreven op "x, y" (regel 680).
De variabele "m", die het aantal beurten telt, wordt in regel 690 met één verhoogd en de computer wordt door regel 700 teruggestuurd naar de start van dat deel van het programma dat we net hebben besproken. Klaar voor de volgende zet!
Het enige dat we nog niet hebben besproken is subroutine 900, die de hindernissen op het scherm plaatst en de regels 800 tot en met 860, waarmee het aantal beurten op het scherm wordt geschreven en de computer vraagt of u nog een spel wil spelen.
Beide delen zijn gemakkelijk te begrijpen.
Wij willen u nog even wijzen op een extra voordeel van het gebruik van de "SCREEN$" voor het onderzoeken of een bepaalde symboolplaats vrij is. Hetzelfde programma-onderdeel zorgt er automatisch voor, dat het sterretje niet uit de doolhof kan ontsnappen. Het labyrint is immers omringd met een rand, en het programma komt in aktie als het sterretje deze rand zou willen overschrijden.
We hoeven dus geen extra "IF"-instructies op te nemen! Na bestudering van dit programma en de technieken in dit hoofdstuk beschreven, moet u zonder meer in staat zijn "PEEK"-, "POKE"- en "SCREEN$"-instructies te gebruiken om er de fraaiste bewegende beelden mee te produceren!
Zo zou u bijvoorbeeld, als oefening met een "SCREEN$"-functie het "squash"-programma kunnen omvormen door het kaatsen van de bal tegen het slaghout of de grenzen van het veld te laten detecteren.

**"Lopende" beelden**

We hebben tot nu toe een indrukwekkend arsenaal aan technieken behandeld, waarmee we afbeeldingen op het scherm kunnen laten bewegen. Eén probleem is hiermee echter niet op te lossen: hoe kunnen we diverse symbolen tegelijkertijd laten bewegen?
In het algemeen gesproken kan men op de Spectrum geen verschillende symbolen laten bewegen als men in BASIC programmeert.
Er is echter één uitzondering.
Probeer het volgende programma eens uit:

```
10 CLS
20 POKE 23692,255
30 PRINT AT 21,RND*31;"*"
40 GOTO 20
```

Door de instructie in regel 30 gaat het scherm "scrollen", dus telkens een regel naar boven bewegen. Hetgeen als optisch gevolg heeft, dat alles wat op het beeld is geschreven of getekend zich vertikaal naar boven beweegt.
De snelheid waarmee dit gebeurt is onafhankelijk van de hoeveelheid informatie op het scherm. Met regel 20 onderdrukken we de

''Scroll?''-boodschap, die de computer normaliter na een aantal scroll's op het scherm schrijft.
We hebben nu een eenvoudig systeempje om alles wat op het scherm staat te laten bewegen, maar hoe laten we iets stil staan?
Het eenvoudigst kunnen we dat doen door na iedere ''scroll'' een nieuwe ''PRINT''-instructie te geven, met steeds dezelfde coördinaten.
Als u het bovenstaande programma uitbreidt met de volgende regel:

```
35 PRINT AT 0,16;"V";
```

ontstaat de basis voor een ''stuur een ruimteschip door de sterren''-spel!

**Een ski-slalom spelletje**

Een mooi voorbeeld van de bruikbaarheid van de beschreven scroll-methode is onderstaand programma, waarmee we een skiër langs een aantal poorten laten laveren.

```
10 RND 0
20 LET b$="<>"
30 CLS
40 DIM c(26,2)
50 GOSUB 5000
60 GOSUB 1000
70 LET t=0
80 LET p=0
90 LET x=16
100 GOSUB 2000

110 GOTO 4000

1000 FOR i=1 TO 25
1010 LET c(i,1)=INT(RND*5)+5
1020 LET c(i,2)=INT(RND*25)+4
1030 NEXT i
1040 LET c(1,1)=21
1050 RETURN

2000 POKE 23692,255:PRINT AT 21,c(1,2);">";
2010 LET k=1
2020 LET j=0
2030 LET m=0
2040 LET i=2
2050 LET s=0
2060 LET s=s+1
2070 LET t=t+1
2080 PRINT AT 21,0''
2090 GOSUB 3000
2100 IF s<>c(i,1) THEN GOTO 2060
2110 POKE 23692,255:PRINT AT 21,c(i,2);b$(j+1);
2120 LET j=NOT j
2130 LET i=i+1
2140 IF i<>26 THEN GOTO 2050
2150 FOR i=1 TO 23
2160 LET t=t+1
2170 POKE 23692,255:PRINT AT 21,0"
2180 GOSUB 3000
2190 NEXT i
2200 RETURN

3000 LET a$=INKEY$
3010 IF a$="5"THEN LET x=x-1
3020 IF a$="8"THEN LET x=x+1
3030 PRINT AT 0,x;"*";
3040 IF t<>c(m+1,1) THEN RETURN
3050 LET t=0
3060 LET m=m+1
3070 LET k=NOT k
3080 IF NOT k AND (x-c(m,2))>0 THEN RETURN
3090 IF k AND (x-c(m,2))<0 THEN RETURN
3100 LET p=p+1
3110 BEEP .05,20:PRINT "h"
3120 RETURN

4000 PRINT
4010 PRINT "U passeerde ";p;" poortjes"
4020 PRINT "Probeert u het nog eens ?";
4030 INPUT a$
4040 PRINT a$
4050 IF a$(1)="n" THEN STOP
4060 IF a$(1)<>"j" THEN GOTO 4020
4070 PRINT "Zelfde parcours ?";
4080 INPUT a$
4090 PRINT a$
4100 IF a$(1)="y" THEN GOTO 70
4110 IF a$(1)<>"n" THEN GOTO 4070
4120 GOTO 60

5000 CLS
5010 PRINT
5020 PRINT TAB 8;"S K I  R U N"
5030 PRINT TAB 8;"--------------"
5040 PRINT
5050 PRINT "U krijgt een afdaling"
5060 PRINT "met 25 poortjes."
5070 PRINT
5080 PRINT "U moet links passeren van"
5090 PRINT "< en rechts van >"
5100 PRINT
5110 PRINT "U kunt links en rechts bewegen"
5120 PRINT "met de pijltoets"
5130 PRINT "(5 en 8)"
5140 PRINT
5150 PRINT "Druk op 'n toets voor nieuwe afdaling"
5160 IF INKEY$="" THEN GOTO 5160
5170 CLS
5180 RETURN
```

Het is belangrijk dat we in regel 20 en overal waar het verder van toepassing is, de combinatie ''kleiner dan'', gevolgd door ''groter dan'' gebruiken, in plaats van het voor de hand liggende ''is niet gelijk aan''-symbool!
Hoewel dit programma in grote lijnen vrij eenvoudig van opzet is, zijn er toch wel wat handigheidjes in verwerkt, die voornamelijk worden gebruikt om de totale scherm-layout in de gaten te houden. Allereerst worden door middel van subroutine 5000 de nodige instructies voor de speler(s) op het scherm geschreven. Nadien bouwen we met subroutines 1000 een willekeurige wedstrijdbaan op, door het genereren van twee willekeurige getallen. Het ene bepaalt de afstand in aantal ''scroll''-bewegingen tussen twee opeenvolgende poorten en het andere legt de horizontale coördinaten van de poorten vast.
Het spel begint met een verwijzing naar subroutine 4000, waarmee de eerste poort op het scherm wordt getekend. Nadien gaat het scherm ''scrollen'', waardoor de poort beweegt in de richting van de skiër, voorgesteld door een ster-

retje. Na een bepaald aantal "scroll"-bewegingen, voorgesteld door c (2,1), wordt de volgende poort getekend.
Omdat de skiër op de eerste lijn van het scherm wordt getekend, zal de eerste poort de skiër na 21 "scroll"-bewegingen bereiken.
Dan wordt de positie van de skiër vergeleken met de plaats van de poort en bepaald of de skiër aan de juiste kant van de poort staat.
Volgens hetzelfde principe wordt bekeken hoeveel poorten de skiër passeert en hoe vaak hij aan de verkeerde kant van een poort zit. De details van dit programma vindt u in de regels 2000 tot en met 3120 en zijn niet moeilijk te begrijpen.

**Conclusie**

Met de technieken die we in dit hoofdstuk hebben beschreven, is het zonder meer mogelijk tamelijk gecompliceerde uitgekiend-geprogrammeerde grafische spelletjes te ontwikkelen.
De mogelijkheden zijn met de uitgewerkte voorbeelden natuurlijk lang niet uitgeput! Zo zou u de "scroll"-techniek kunnen combineren met de "SCREEN$"-functie of gaan "PEEK"-en en "POKE"-en in het attributengeheugen, waardoor gekleurde bewegende objekten ontstaan. Toch zal er een moment komen, waarop u zult moeten erkennen dat programmeren in BASIC te traag is om tegemoet te komen aan de steeds toenemende eisen die u aan grafische programma's gaat stellen.
Er is dan maar één oplossing: BASIC laten voor wat het was en overschakelen op programmeren in machine-taal.
Maar dát is een heel ander verhaal.